A Probabilistic Approach for the System-Level Design of Multi-ASIP Platforms

Micconi, Laura

Publication date:
2015

Document Version
Publisher's PDF, also known as Version of record

A Probabilistic Approach for the System-Level Design of Multi-ASIP Platforms

Laura Micconi

Kongens Lyngby 2014
PhD-2014-347

Technical University of Denmark
Department of Applied Mathematics and Computer Science
Building 324, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253031
compute@compute.dtu.dk
www.compute.dtu.dk PhD-2014-347

Summary (English)

Application Specific Instruction-set Processors (ASIPs) offer a good trade off between performance and flexibility when compared to general purpose processors or ASICs. Additionally, multiple ASIPs can be included in a single platform and they allow the generation of customized heterogeneous MPSoC with a relatively short time-to-market. While there are several commercial tools for the design of a single ASIP, there is still a lack of automation in the design of multi-ASIP platforms.

In this thesis we consider multi-ASIP platforms for real-time applications. Each ASIP is designed to run a specific group of tasks that we identifies as a *task cluster*. With real-time applications, to decide how the tasks should be clustered, we perform a schedulability analysis of the system to verify if the deadlines of the applications can be met. However, to run a schedulability analysis, we need to know the WCET of each task that is available only after an ASIP is designed. Therefore, there is a circular dependency between the definition of the task clusters and the impossibility of defining them without knowing the WCET of the tasks as the ASIPs have not been defined yet.

Many approaches available in the literature break this circular dependency considering pre-defined task clusters or considering a small set of micro-architecture configurations for each ASIP. We propose an alternative approach that uses a probabilistic model to consider the design space of all possible micro-architecture configurations. We introduce a system-level Design Space Exploration (DSE) for the very early phases of the design that automatizes part of the multi-ASIP design flow. Our DSE is responsible for assigning the tasks to the different ASIPs exploring different platform alternatives. We perform a schedulability analysis for each solution to determine which one has the highest chances of meeting the deadlines of the applications and that should be considered in the next stages of the multi-ASIP design flow.

Summary (Danish)

Applikations specifikke instruktionssæt processorer (ASIPs) tilbyder en god afvejning mellem ydeevne og fleksibilitet i forhold til generelle processorer og dedikerede hardware acceleratorer (ASICs). Flere ASIPs kan inkluderes i en enkelt platform, hvilket tillader realisering af applikations tilpassede heterogen multiprocessor systemer med en relativ kort time-to-market. Mens der er flere kommercielle værktøjer til design og konfigurering af en enkelt ASIP, er der stadig mangel påautomatiseret design af multi-ASIP platforme. Denne afhandling fokuserer påmulti-ASIP platforme til realtids applikationer. Hver ASIP er designet til at køre en specifik gruppe af opgaver. For at afgøre hvordan de enkelte opgaver i realtids applikationen skal fordeles påde forskellige ASIPs, udføres en schedulerbarheds analyse af systemet for at kontrollere, at tidsfristerne kan imødekommes. Men for at kunne køre en schedulerbarheds analyse, er det nødvendigt til at kende den øvre grænse for de enkelte opgavers eksekveringstid (WCET), efter at ASIP'en er konstrueret. Derfor er der en cirkulær afhængighed mellem identifikationen af grupper af opgaver for den enkelte ASIP, som kun kan foretages når WCET kendes, dvs. når ASIP'en er fuldt konfigureret, og det faktum at den korrekte konfigurering først kan bestemmes når der vides hvilke opgaver der skal afvikles påASIP'en.

Eksisterende metoder bryde denne cirkulær afhængighed ved at bruge foruddefinerede grupper af opgaver eller ved kun at betragte et lille sæt af de mulige ASIP konfigureringer. Denne afhandling foreslår en alternativ tilgang, der bruger en probabilistisk model til at repræsentere alle mulige ASIP konfigurationer. Det gør det muligt at automatisere metoden til udforskning af løsningsrummet for en multi-ASIP løsning i de meget tidlige faser af designprocessen. Metoden udforsker forskellige alternative multi-ASIP platforme, ved gennem en statistisk analyse at finde den tildeling af opgaver til de enkelte ASIPs, der giver den højeste sandsynlighed for at opnåen løsning der lever op til kravene, efter en fuld implementering er gennemført.

Preface

This thesis was prepared at the Department of Applied Mathematics and Computer Science at the Technical University of Denmark in fulfillment of the requirements of the Ph.D. program. The Ph.D. program was partially funded by ASAM, a project in the framework of the European ARTEMIS Research Program and ARTEMIS Joint Undertaking.

This work was supervised by Professor Jan Madsen and Associate Professor Paul Pop and conducted occasionally in collaboration with Dr. Deepak Gangadharan.

The thesis describes a system-level design space exploration for the design of multi-ASIP platforms, i.e. MPSoC that contains multiple Application-Specific Instruction set Processors. The approach described is based on a probabilistic performance model that finds its application in the very early phases of the multi-ASIP design flow. The thesis consists of nine chapters (including Introduction and Conclusion) and two appendices.

The thesis does not contain any material that has been accepted for the award of any other degree or diploma in my name, in any university or other institution and, to the best of my knowledge does not contain any material previously published by another person, except where due reference is made in the text of the thesis.

Part of the experimental evaluation presented in this thesis has been done using Silicon Hive (now Intel Benelux B.V.) technology and tools. The results obtained using Intel's tools should not be used in any way as a reference to evaluate Intel technology or

to compare Intel's tools with other commercial or research tools, as only a subset of the functionalities and optimization offered by the tools has been used and/or made available under our University license agreement.

Lyngby, July 2014

Laura Micconi

Publications and technical reports

Peer reviewed publications

1. Jozwiak, L.; Lindwer, M.; Corvino, R.; Meloni, P.; **Micconi, L.**; Madsen, J.; Diken, E.; Gangadharan, D.; Jordans, R.; Pomata, S.; Pop, P.; Tuveri, G.; Raffo, L. and Notarangelo, G.: "ASAM: Automatic architecture synthesis and application mapping". In Microprocessors and Microsystems, Volume 37, Issue 8, Part C, November 2013, Pages 1002-1019.

2. Gangadharan, D.; **Micconi, L.**; Pop, P. and Madsen, J.: "Multi-ASIP Platform Synthesis for Event-Triggered Applications with Cost/Performance Trade-offs". In IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, Taipei, Taiwan, 2013.

3. **Micconi, L.**; Gangadharan, D.; Pop, P. and Madsen, J.: "Multi-ASIP Platform Synthesis for Real-Time Applications". In Proceedings of SIES 2013 - 8th IEEE International Symposium on Industrial Embedded Systems, Porto, Portugal, June 2013.

4. **Micconi, L.**; Corvino, R.; Gangadharan, D.; Madsen, J.; Pop, P. and Jóźwiak, L.: "Hierarchical DSE for multi-ASIP platforms". In ECyPS 2013 - EUROMICRO/IEEE Workshop on Embedded and Cyber-Physical Systems, pages 50-53, Budva, Montenegro, 2013 *[Received best student paper award].*

5. Jóźwiak, L.; Lindwer, M.; Corvino, R.; Meloni, P.; **Micconi, L.**; Madsen, J.; Diken, E.; Gangadharan, D.; Jordans, R.; Pomata, S.; Pop, P.; Tuveri, G. and

Raffo, L.: "ASAM: Automatic Architecture Synthesis and Application Mapping". In Proceedings of DSD 2012 - 15th Euromicro Conference on Digital System Design, pages 216-225, Cesme, Izmir, Turkey, 2012.

Miscellaneous

Micconi, L.: "Multi-ASIP Platform Synthesis for Real-Time Applications". Student research abstract (SRC) competition at SAC 2013 – 28th Symposium on Applied Computing. Coimbra, Portugal, March 2013 *[1st price winner].*

Technical reports

1. **Micconi, L.**; Gangadharan, D.; Madsen, J. and Pop, P.: "Demonstrator of DSE tool". Deliverable 2.7, ASAM Project, November 2013. *[Set of videos, Public]*
2. **Micconi, L.**; Gangadharan, D.; Madsen, J. and Pop, P.: "Prototype tools for design space exploration". Deliverable 2.6, ASAM Project, January 2013. *[Restricted]*
3. **Micconi, L.**; Boesen, M. R.; Gangadharan, D.; Madsen, J. and Pop, P.: "Prototype tools for system analysis". Deliverable 2.5, ASAM Project, April 2012. *[Restricted]*
4. **Micconi, L.**; Madsen, J. and Boesen, M. R.: "Method of and report on system analysis". Deliverable 2.3, ASAM Project, November, 2011. *[Public]*
5. **Micconi, L.**; Madsen, J.; Pop, P.; Kienhuis, B.; Corvino, R. and Jóźwiak, L.: "Report on initial version of the hierarchical application model". Deliverable 2.2, ASAM Project, May, 2011. *[Public]*
6. **Micconi, L.**; Iordache, G.; Madsen, J.; Pop, P.; Meloni, P.; Lindwer, M.; Cocco, M.; Notarangelo, G. and Guidetti, E.: "Initial version of generic platform model". Deliverable 2.1, ASAM Project, January, 2011. *[Public]*

Under review

Micconi, L.; Madsen, J. and Pop, P.: "An Uncertainty Model for System-Level design of Multi-ASIP Platforms". Under review at *Integration, the VLSI Journal*, Special Issue On Application and Domain-Specific Computing.

Acknowledgements

I would like to express my special appreciation to my advisors Professor Jan Madsen and Associate Professor Paul Pop. A special thank to Jan for his precious suggestions, guidance and support when needed; he is always thinking ahead and every discussion with him has been very enlightening and stimulating. I would like to thank Paul that has helped me with a methodic and disciplined critical thinking. He has contributed in improving the formulation of the ideas and his scrutiny of my technical writing has been invaluable. Without Jan's and Paul's supervision and constant help this dissertation would have not been possible.

Moreover, my gratitude goes to Dr. Deepak Gangadharan that with helpful comments and guidance has taught me how to approach and investigate a research problem.

I would also like to thank my committee members, Professor Jens Sparsø, Professor Ingo Sander and Professor Andrew David Pimentel for reviewing the thesis and providing invaluable comments and suggestions during the defense.

My appreciation goes to all the partners of the ASAM project; it has been extremely stimulating and motivating to work with them. I learned a lot from all of them, especially from Dr. Rosilde Corvino. I will always remember our Skype meeting for discussing the project; Rosilde has always been available with suggestions and comments and has been there when a good laugh was needed in the critical phases of the project. I would like to thanks also Dr. Menno Lindwer that supervised and guided me during the external stay period at Intel Benelux, I will never forget those months in which I had an insight of the industrial world in a stimulating and innovative environment such as Intel.

A giant thank to my colleagues from embedded system engineering (ESE) group at DTU, especially Alessio, Alex, Aske, Davide, Domi, Eduardo, Fontas, George, Massimo (with Valette), Sahar, Valia and Wolfgang. I will always have good memories of the lunches and of the breaks with the coffee-crew and the best coffee of whole DTU. My time at DTU has been great especially thanks to them. I need to thank Alessio, Alex, Domi and Sahar also for reviewing part of the thesis and for their precious feedback. I will miss the great fun I had while biking with Alessio and during the tango classes with Valia. A big thank to Karin, she has always been very kind, willing to help and answer all my questions. Thanks to her I always felt very welcomed in the ESE group. Thanks also to Giovanni, Gosia and Paolo; I have great memories with them: I love their company and the dinners at their place.

My immense appreciation and thanks go to German, Letizia and Eduard. German has always been there for me, he has been my reference point before, during and after the PhD studies; he has given me precious suggestions and guidance and he has also offered me affection and a sincere friendship; I have always felt loved, I really learned a lot from him. Bella Leti has been the person that cheered me up when needed. I feel so lucky that I met her at the beginning of my PhD studies; she is the most generous, understanding and fun friend that someone can hope for. I love each one of our conversations and Skype-vinate. Eduard has been my main reference point during the external stay period, I loved cooking and chatting with him, he has been my happy thought in Eindhoven.

Finally, but not least, I want to thank my pillars of strength during these years: my parents. They have always given me constant support through the ups and downs of my academic career and they have always been there for me when needed, I have never felt alone. I cannot imagine being the person I am today without such a great mum and dad. And a giant thank to my grandma that has supported me even if this means living far away from home.

Thanks to everyone that helped me get to this day.

Contents

List of Figures

List of Tables

Listings

Abbreviations

API application programming interface.

ASAM Automatic Architecture Synthesis and Application Mapping.

ASAP as soon as possible.

ASIC Application Specific Integrated Circuit.

ASIP Application Specific Instruction-set Processor.

CDF cumulative distribution function.

CE communication element.

DFG data flow graph.

DSE Design Space Exploration.

DSP Digital Signal Processor.

ET event-triggered.

FFT fast fourier transform.

fpps fixed priority preemptive scheduling.

fps frame-per-second.

FU functional unit.

GPP General-Purpose Processor.

GUI graphical user interface.

HDL hardware description language.

ILP instruction level parallelism.

IP intellectual property.

ISA instruction-set architecture.

KPN Kahn process network.

MCS Monte Carlo simulation.

MEM memory.

MPSoC Multi-Processor System-on-Chip.

NI network interface.

NoC Network-on-Chip.

PA Pipeline Analysis.

PDF probability density function.

PE processing element.

RF register file.

RTA response time analysis.

SDFG synchronous dataflow graph.

SFM straightforward method.

SH Silicon Hive.

SoC System-on-Chip.

SSEA Steady-State Evolutionary Algorithm.

TDM time-division multiplexing.

TG task graph.

TLA Task-Level Analysis.

UM WCET Uncertainty Model.

VEX VLIW Example.

VLIW Very Long Instruction Word.

WCET worst-case execution time.

CHAPTER 1

Introduction

Today embedded systems include an increasing number of processors as an answer to higher performance and energy efficiency requirements; they are currently used for executing a wide variety of real time applications from the automotive, multimedia and networking domains. Multi-Processor System-on-Chip (MPSoC), especially when used in portable devices, have tight area, power and performance constraints that are given by the applications they implement. This increases the complexity of the design of the system which key design constraints are flexibility and performance. General-Purpose Processors (GPPs) are flexible platforms and run applications from various domains, but they fall behind on performance in comparison to ASICs. On the other hand, ASICs are designed to run specific applications and therefore lack flexibility. Application Specific Instruction-set Processors (ASIPs) combine the best of both worlds by incorporating application specific custom instructions, thereby giving more flexibility than ASICs and better performance than GPPs. ASIPs are designed such that they are optimized to run a specific set of functions.

Recently, an increasing number of ASIPs is used in embedded platforms, together with heterogeneous processing elements (PEs), for the implementation of real-time systems (especially image/video processing systems) [21, 83, 41]. Therefore, there is an increased tendency in designing multi-ASIP platforms, i.e. heterogeneous platforms that may contain multiple ASIPs.

Many companies are providing tools for the design of single ASIPs; among them Ca-

dence with the Tensilica toolchain for the XTensa processor [95] and Synopsys with the *Processor Designer tool* (using Lisa language) [92]. Additionally, ASIPs are now used in commercial processors, e.g., the Intel Atom Clovertrail+, a SoC for smartphones, that includes an ASIP for image signal processing [67].

In this thesis, we focus on the system-level (or macro-architectural level) design of multi-ASIP platforms. This chapter is organized as follows. First, we present the characteristics of an ASIP and the design flows that can be used for the implementation of a single ASIP (Section 1.1). Second, we described the motivations behind the inclusion of multiple ASIPs in a platform and the challenges in the design of a multi-ASIP platform (Sections 1.2 and 1.3). Then we present the related work (Section 1.4), the objective of this thesis (Section 1.5), our contributions (Section 1.6) and the structure of the thesis (Section 1.7).

1.1 Why ASIPs?

1.1.1 Application-specific instruction set processors

An ASIP is a programmable processor that is optimized for a particular application. ASIPs can be used as *custom processors* or *programmable accelerators*. They can be used to satisfy the need of modern embedded systems of having multiple functionalities integrated on a single System-on-Chip (SoC) [92]. The design of an ASIP requires the definition of the processor architecture (i.e. micro-architecture and instruction set) and the development of the corresponding simulator and software toolchain (compiler, assembler and linker). The micro-architectural description of an ASIP is usually defined starting from its instruction-set architecture, ISA (as described in Section 1.1.2). ASIPs provide a good trade-off between performance and power efficiency. In Figure 1.1, an intuitive comparison between ASIPs and other families of processors is shown.

ASICs are highly power efficient, but there are multiple factors that make ASICs design and manufacturing expensive [68], such as the high non-recurring engineering cost and the long time-to-market. The non-recurring engineering costs for ASICs increase together with the transistor density and the use of sub-micron technology. The mask-set can exceed one million dollars and the development of new tools for ASIC design demands a high cost as well [16]. Using ASIPs, more flexible designs can be generated and the non-recurring design costs can be reduced: ASIPs can be configured to run multiple applications and, vice versa, a design can be reused across multiple applications; this allows producing higher volumes of processors and amortizing design and manufacturing costs [68]. At the same time, the time-to-market is reduced.

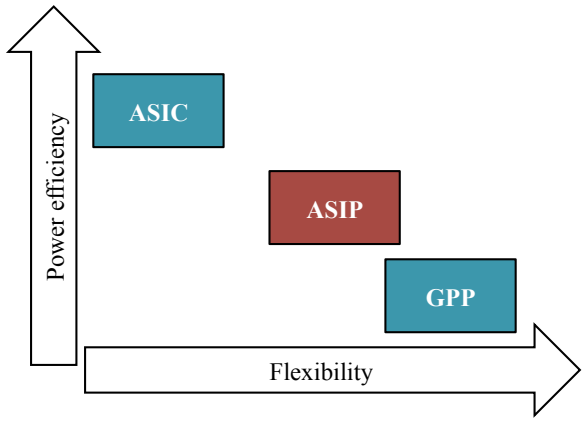


Figure 1.1: Flexibility-power trade-off of different architectural options [12]

Compared to GPPs, ASIPs offer an energy efficient alternative still providing good performance; for example, they can be used in all portable devices that are targeting specific applications and where a GPP cannot be used due to its inefficient power management.

Additionally, an ASIP designer can count on tools for the automatic generation of the software toolchain and evaluation of their designs (as presented in Section 1.1.2). As a result, ASIPs can satisfy the requirements of those designers seeking high-performance and low power with the advantages of an automated design methodology [16].

1.1.2 ASIP design

Figure 1.2 highlights the generic steps for the design of a single ASIP, as described in [40]. An ASIP is designed according to the application it has to run and the input constraints (e.g. performance, cost); hence, the first step is the analysis/profiling of the application code. A common approach is to define the micro-architecture starting from an ASIP template (selected from a library). The definition of a specific ASIP micro-architecture includes then a micro-architecture DSE for the identification of the appropriate number and type of functional units, memory, issue slot, etc., required by the application. After an initial micro-architecture is defined, the instruction set is generated, which can include custom instructions to speed up the execution.

There are multiple tools for the design of a single ASIP. Examples of ASIP designs are described in [46, 73, 79, 76, 101, 33, 45, 43]. In particular, [46, 73, 79, 76] employ the LISATek tool (now part of Synopsis [92]), [101] applies Target Compiler Tech-

nologies's tools (now also part of Synopsis [92]), [33] uses the Tensilica AutoTIE tool, [45] describes NoGAP and [43] uses the Silicon Hive (SH) toolchain. Following, we provide a brief description of the design flow proposed by these tools.

The LISA description language speeds up the design flow in Figure 1.2 by supporting the automatic generation of the software toolchain (retargetable compiler, assembler and linker), of the simulator and of a synthesizable HDL code, out of a processor description [76]. The processor description contains the instruction-set architecture (ISA) and the behavioral and timing model of the processor. Additionally, LISA is inserted into the LISATek tool suite that provides support for the design space exploration of the architecture. The simulator and synthesized HDL description can be used to evaluate the performance and cost of a LISA processor description and adjust it to the designer's requirements. The authors in [76] use the LISATek tool for the creation of two differently optimized ASIPs for an FFT algorithm. The development of the two ASIPs required three man weeks.

Target Compiler Technologies provide a tool suite called IP Designer to support the design of ASIPs. Each processor is described using nML description language which captures the structural characteristics of the design (i.e., the datapath architecture) and the instruction set architecture. IP Designer implements an entire retargetable toolchain for the ASIP design, a simulator and an HDL generator. In [101], the authors use IP Designer for the design of a ultra-low-power ASIP for an Electrocardiogram (ECG) application with software and hardware optimization to improve performance and minimize the power consumption.

A Tensilica processor [95] is described using the TIE (Tensilica Instruction Extension) language that specifies the ISA and its extensions. Additionally, TIE allows having the ISA extension implemented in hardware and have them recognized by the Tensilica software toolchain. Tensilica AutoTIE described in [33] is a tool for the automatic extension of the base ASIP's ISA with Very Long Instruction Word (VLIW) instructions, vector operations and custom operations. AutoTIE explores a large number of ISA extensions (based on the profile of the loops of the input application) and then uses performance and area estimations to evaluate the combinations of multiple extensions and to generate an ASIP.

NoGAP [45] is a research tool that aims at the automatic generation of a software toolchain for ASIPs. The tool provides assembler, simulator and a hardware generator. NoGAP provides the designer with a very flexible architecture and does not use an initial ISA template. However, NoGAP does not provide an automatic DSE; the designer is responsible for the definition of the initial architecture that he considers compatible with the application or application domain. In [45], NoGAP is used to generate a single issue RISC processor with DSP extensions. The development required three man weeks, but considering that an expert designer is involved in the architecture definition.

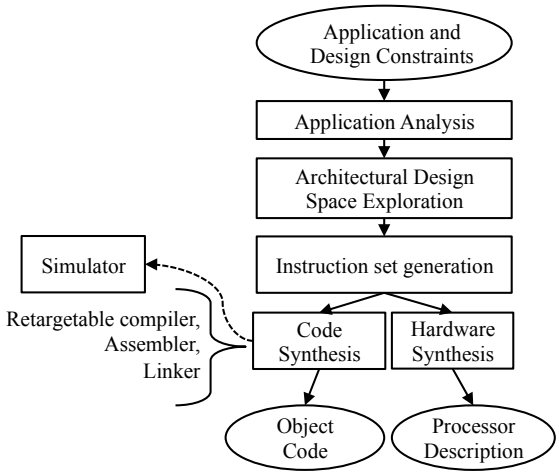


Figure 1.2: Example of ASIP micro-architecture synthesis flow [40]

SH tools constitute a toolchain for the development of VLIW ASIPs [62]. SH offers proprietary languages for the hardware description of the processors and of the multi-processor platform. Additionally, it provides a retargetable compiler and a simulator for the execution of the application C code on the user-defined platform. In this section, we do not provide additional details about SH tools that are thoroughly described in Section 7.2.4. We use SH tools for the experimental evaluation of the design approach described in this thesis. Silicon Hive is now part of Intel Benelux B.V.

1.2 Why a multi-ASIP platform?

In the previous section, we mentioned the advantages derived from employing ASIPs. In this section, we look into reasons for using multi-ASIP platforms, i.e. MPSoCs containing multiple ASIPs.

Future embedded systems will demand higher computational efficiency and cheap implementations. ASIPs represent an attractive solution especially in areas such as multimedia, networking, and signal processing [68]. As an example, let us consider the multiple applications for image processing running on today mobile phones. These applications require specialized hardware for efficient execution; however, as the number of different applications is growing, also the number of specialized PEs for their execution has to increase. Integrating multiple ASIPs in a single SoC is a promising solution with high flexibility; it allows having customized PEs to run applications (or part of

them) and minimizing the energy consumption differentiating between the architecture implemented by each ASIP. Having multiple heterogeneous ASIPs, we can partition the applications into tasks, assign the tasks to the different PEs and exploit the task level and pipeline parallelism, intrinsic to signal processing and multimedia domains.

1.3 Challenges in multi-ASIP design

In Section 1.1.2 we described possible design methodologies for the design of a single ASIP. An ASIP is designed according to the application(s) that it has to execute.

However, when we consider multiple ASIPs and multiple input applications or a single input application partitioned into multiple tasks, we add an extra degree of complexity to the design problem. In fact, we need to establish how these tasks are assigned to the different ASIPs that are then designed according to the tasks they have to run. This task assignment is not straightforward, as it involves inter-dependent decisions on macro-architecture (or system-level) and micro-architecture (i.e. ASIP) levels. These decisions affect the number of ASIPs, their micro-architecture and interconnection together with the assignment of tasks to the different ASIPs; the optimization of the multi-ASIP platform is, therefore, a NP-complete problem [29].

We call the assignment of tasks to the different ASIPs *task clustering*. Each group of tasks is a *task cluster* and corresponds to a single ASIP. In order to determine the task clustering for one or more applications, we need to perform a schedulability analysis that requires information about the execution performance of the tasks and of the data dependencies among them. It is possible to know the performance of each task when an ASIP is implemented/designed.
However, the ASIP design approaches described in Section 1.1.2 imply that an ASIP is implemented only after knowing which tasks it has to run.
Therefore, we have a tight connection between the task clustering and the design of each ASIP as they depend on each other.

Depending on the number of tasks and ASIPs included in the platform, a very large number of task clusters has to be evaluated during DSE. For each task cluster, the micro-architecture design of a *single* ASIP involves a number of steps and the design of a single ASIP can requires multiple days (Section 1.1.2). Further, the design space of a ASIP micro-architecture is very large and depends on the number and data widths of register files (RFs) and memory blocks (MEM), and number of functional units (FUs) (Figure 1.3). Hence, platform design with multiple ASIPs is non-trivial as it needs to take the design space of the ASIP micro-architecture into consideration when exploring various platform solutions. In the rest of the thesis, we use the term *micro-architectural configuration* to indicate the micro-architecture resulting from a specific ASIP design.

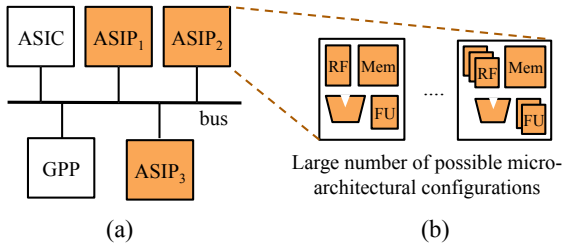


Figure 1.3: Example of micro-architecture configurations for an ASIP [69]

Consider an example with three tasks τ_1, τ_2 and τ_3 for which a multi-ASIP bus-based platform has to be designed. We use the worst-case execution time (WCET) as the performance value for the tasks. We define a *task cluster* tc_k, as the set of tasks used at the input of the flow in Figure 1.2 to design an ASIP PE_k. Let us now consider two different ways of clustering the three tasks: case (a) and (b). In case (a), tasks are assigned to three different ASIPs, PE_1, PE_2 and PE_3, with the corresponding clusters $tc_1 = \{\tau_1\}$, $tc_2 = \{\tau_2\}$ and $tc_3 = \{\tau_3\}$. The single ASIP design flow (Figure 1.2) can be followed and each ASIP micro-architecture can be optimized to satisfy the particular task. Now that we have each ASIP defined for case (a), we can determine the worst-case execution time (WCET) of each task. Let us denote by $C_j^{PE_k}$, the WCET of task τ_j on ASIP PE_k. Although an ASIP PE_k has been designed and tuned for a certain set of tasks tc_k, it could also run a task $\tau_j \notin tc_k$ (if binaries are compatible). Let us consider that we run τ_2 on PE_1, tuned for τ_1. The micro-architecture of PE_1 has not been specifically tuned for the functionality of τ_2, as a consequence we can expect that $C_2^{PE_2} \leq C_2^{PE_1}$ (note that the increase in $C_2^{PE_1}$ also depends on the similarity between τ_1 and τ_2) (this example is depicted in Figure 1.4).

In case (b), τ_1 is clustered with τ_2 such that $tc_1' = \{\tau_1, \tau_2\}$ and $tc_2' = \{\tau_3\}$. Since the task clustering has changed, we need to re-design all the ASIPs from case (a), otherwise the WCET of τ_2 may increase too much. This will again impact the WCETs of the tasks. In our case, the WCET of τ_2 will decrease and that of τ_1 might increase, depending on how much the ASIP design will satisfy the functionality required by one task compared to the other.

This example shows that every time a task clustering changes, the WCETs will change. For every WCET change, we need to evaluate again the schedulability of the applications. However, we cannot know the WCET of a task before we have designed the corresponding ASIP. Therefore, after each re-clustering decision we would have to run a complete ASIP design flow, as described in Section 1.1.2, for each affected ASIP. As mentioned, an ASIP design takes days, so it cannot be done during the DSE at the platform level. Hence, to perform DSE during platform design, we need the WCETs to evaluate the schedulability, and WCETs can only be known after the platform has

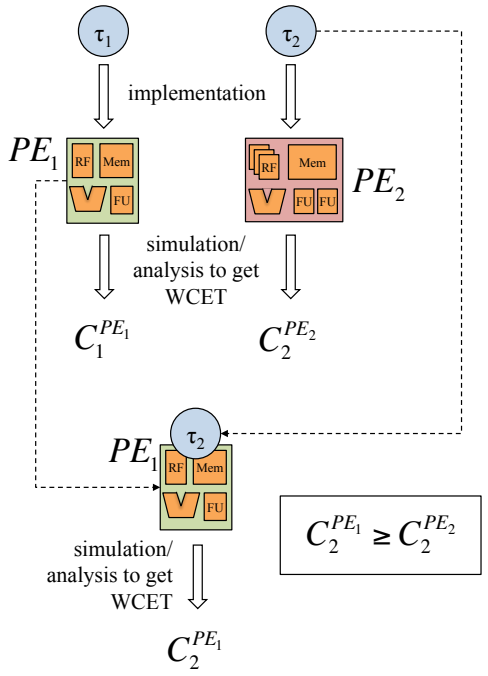


Figure 1.4: Effect of a task clustering on the WCET

been fully designed. This *circular dependency* drastically limits the number of platform alternatives that can be considered during DSE.

Possible options to break this circular dependency, include having one ASIP for each task or a single ASIP for all tasks. Then, the designer will have to split or merge the tasks to get to feasible implementations that meet the performance requirements and do not exceed the platform cost (e.g. the total area of the platform). However, these approaches require the work of an experienced designer. Another possible alternative that is adopted in many design methods described in literature (Section 1.4), is to consider only a small set of predefined micro-architecture configurations for each ASIP so that the design flow of a multi-ASIP platform falls back into a classic MPSoC design. In this case the risk is to ignore potentially good solutions as the micro-architecture DSE is limited beforehand.

*The goal of this thesis is to offer an alternative approach to break this **circular dependency**, considering the space of all possible micro-architecture configurations, and to define a multi-ASIP platform given one or more applications and its constraints as input.*

1.4 Related work

There has been a significant effort in research for supporting the design and evaluation of MPSoCs. There are design approaches that target the optimization of a platform implementation including the number and type of processors, their interconnections and the memory sizes. In other approaches the platform is already available and the focus is on the performance evaluation of different mapping and scheduling. Application partitioning is also considered. Some work proposes general design approaches that can be applied to a wide variety of application domains. For example, the Platform-based design paradigm [82, 49] in which a design is defined by a *meeting-in-the-middle* process: the specifications of the system to implement are gradually refined and matched with possible *platforms*[1] at different level of abstractions. The work in [47] proposes a simulation frameworks for accurate performance estimation of applications executing on multi-threaded multi-processor platform. The authors adopt separate models for the application and the platform that are combined during the DSE of mappings. They also consider a temporal (allocation of time budget) mapping to capture multi-threading. However, the platform composition is well known and therefore an accurate model of the performance/cost of each processing and communication architecture is available.
In a similar manner, in [98, 24], the authors assume that the platform is already available (predefined PEs and communication architecture) and they focus on the exploration, using ant colony optimization, of different partitionings of the application considering the data dependencies of the tasks. In this thesis, we provide a technique for the evaluation of different task clustering solutions that is also used to guide the platform definition.

There is then a large body of work for the design of application and domain specific MPSoCs, i.e design methods that produce multi-core platforms targeting specific applications. Many of these works do not use ASIPs [66, 9, 44, 52, 11, 78, 54, 23, 57].
[66, 9] adopt a high level description of the application and initial platform to define the number and type of processors to use (taken from a library), the number of I/O ports of each PE, the size of the memories (dedicated and shared memories) and the interconnection according to an application mapping established by the designer. However, the authors do not consider the effect of different task clusterings or the potential of using ASIPs.
In [44, 52], the authors use UML to model the application, the platform and the design constraints; they explore different macro-architecture possibilities but always considering PEs taken from a library with well known performance and cost.
The approaches in [11, 78, 54, 23, 57] target multimedia or signal processing applications. In [11], MP-ARM, a SystemC simulation engine for MPSoC has been developed. The simulator supposes an AMBA-like bus as communication architecture and ARM CPU as PEs. Although the authors explore different arbitration policies, cache sizes and

[1] a platform is a library of components that can be assembled to generate a design at that abstraction level

the effect of different application partitioning, they consider a fixed macro-architecture composition with predefined PEs and do not perform mapping or scheduling DSE.
Daedalus is described in [78]; it is an automated design environment for for system-level architectural exploration, system-level synthesis, programming, and prototyping on FPGA. It adopts Sesame [23] for the system-level modeling and simulation. Daedalus builds an MPSoC using the PEs from a library and interconnects them using a crossbar switch or a shared bus [77] given a Kahn process network (KPN) model of the input application. This design flow allows the generation of multi-processor systems targeting multimedia or signal processing applications; it optimizes the processing and communication element, defining a customization at platform level, but does not take into account the customization of the micro-architecture of each PE that is one of the goals of this thesis.
Sesame [23] follows a Y-chart approach (Figure 1.5a) with separated models for the applications, modeled as a KPN, and the platform. Application and platform are combined and evaluated during mapping; its multi-objective DSE can be used in the early phases of the design to guide the designer towards the most promising solutions. The work presented in this thesis has some similarities with Sesame: we also propose a method to be applied in the very early phases of the design and that is based on a Y-chart model (Figure 1.5b). On the other hand, we are focusing on ASIPs not yet synthesized without a precise performance and area model for each task/processor as the one available in Sesame. Therefore, we have an additional level of complexity in our design due to the unknown micro-architecture of the PEs.
Another design approach for the definition of an MPSoC platform for a specific ap-

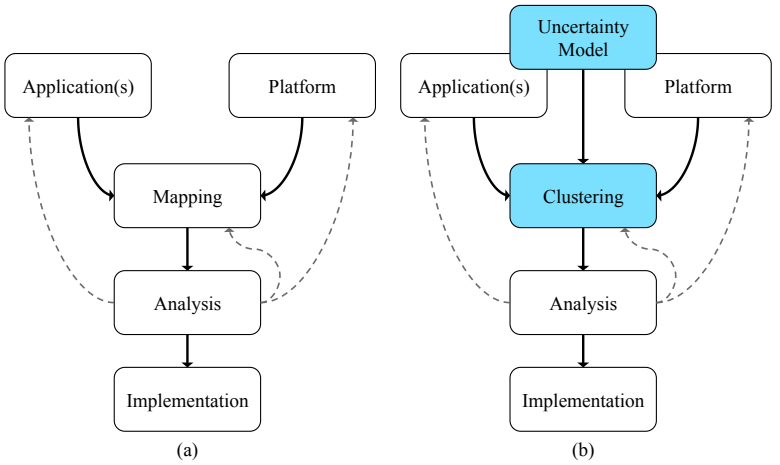


Figure 1.5: Y-chart model for MPSoC design (a) and Y-chart model with Uncertainty Model (UM) for multi-ASIP design (b)

plication is presented in [57] where DSE is used to select the PEs, partitioning the application, modeled as a task graph, and scheduling it. It also runs an optimization

for the communication architecture using a hierarchy of buses. The DSE is guided by a greedy exploration algorithm defined by the authors. The optimization objectives are the platform cost and the performance of the application. As for the other works previously mentioned, the platform is built out of components from a library, whose behavior and performance are well known.
In [54], the MAMPS approach for MPSoC synthesis is described. The authors present a technique that considers multiple use-cases (or scenarios) of execution, in which there are multiple input applications active at different instant of time, and design a custom platform, in which multiple cores are interconnected through FIFOs. The applications are modeled as synchronous dataflow graphs (SDFGs). The authors do not explore multiple mapping solutions (there is an arbitrary assignment of the SDFG actors to the PEs) or interconnections type; as in the previous cases, a set of already synthesized core is available (e.g. Xilinx Microblaze).

Finally, there are approaches that are considering multiple ASIPs [74, 14, 27, 85, 41, 99, 6]. Most of these approaches [74, 14, 27] assume that the ASIPs have already been synthesized, whereas in [85, 41] a small set of micro-architectural configurations is considered. In [85], the authors focus on pipelined multi-ASIP systems and explore different task graph partitionings, but they select the processors to include in the platform from a library of pre-configured ASIPs. Moreover, they limit the interconnection network to a set of FIFO queues. Hence, these approaches severely limit the design space, disregarding possibly good solutions because they do not take into account the ASIP micro-architecture design space during platform design.
There are design approaches in which both ASIPs and their interconnections are optimized [99]. However the ASIPs are synthesized starting from a template micro-architecture and the application is arbitrarily partitioned by the designer among a predefined number of ASIPs. In this approach, the authors do no evaluate the possible advantages of the different task clusterings that can also lead to a different interconnection selection.
On the other hand, different task clustering solutions for the definition of a multi-ASIP platform are considered in [6]. The authors propose a systematic methodology for exploring different clustering considering applications that can work in different scenarios (e.g. different compression type for MPEG2 and MPEG4). A platform template generated for a specific application domain is provided as input. The design parameters that can be configured at platform level are the number of processors, external memory size and the bus width; at processor level, the configurable parameters are the memory size, the word size and the number and type of functional units. The task clustering is systematically defined considering the profile information of each task (e.g. task complexity and number of repetitions of each task). Once each cluster of tasks is defined, the processor parameters are defined. The optimization parameters are the area and the energy consumption. However in this approach there is no analysis of the performance of the application executing on the system. There is an evaluation of the latency of each task cluster based on the number of pipeline stages of each processor, but this cannot guarantee that the selected task clustering is the best one for meeting the application

deadline. Additionally, there are no assumptions on scheduling policies and on how to consider the data dependencies among tasks associated with different processors.

None of the approaches previously described addresses the *circular dependency* between the ASIP micro-architecture and WCET (or performance) values (described in Section 1.3). Usually this issue is bypassed by considering that the ASIPs or a limited set of micro-architectural configurations of the ASIPs are given, but this potentially disregards valid solutions. Therefore, to use these approaches, the designer needs to have some experience and knowledge of the type of system that he expects as output.

1.5 Objective

In this thesis we are interested in the design of multi-ASIP platforms customized for the execution of real-time applications. We are targeting platform containing multiple ASIPs, but other types of PEs, as GPPs or ASICs might also be included. We assume be given one or multiple applications with their design constraints as input.

The focus of this thesis is on the macro-architecture design of the platform, i.e. the number of ASIPs, how they are interconnected and how tasks should be clustered on the different PEs to meet the input constraints. We also want to consider the data dependencies among tasks and the influence of the communication time. In particular, we are interested in providing a platform definition that satisfies the performance constraints of the application.

Our goal is to break the *circular dependency* described in Section 1.3 that is an obstacle to the evaluation of different task clustering solutions: a task clustering is evaluated through a schedulability analysis that requires the WCET of each task associated with an ASIP. However, the WCET can be estimated only knowing the micro-architecture configuration of the ASIP, which is not available until a task clustering is established (as each ASIP micro-architecture is configured based on the associated task cluster). The method described should be applied in the very early phases of the multi-ASIP design flow when there is no information available about the platform implementation and about the micro-architecture of each ASIP. Additionally we want a macro-architecture DSE that can be integrated within a complete flow for the synthesis of multi-ASIP platform, in particular the ASAM design flow [43] that we present in Chapter 7.

1.6 Contributions

The main contribution of this thesis is a technique for the definition of a multi-ASIP platform given one or multiple applications as input: we introduce a system-level (or macro-architecture level) DSE that is in charge of assigning the tasks to the different ASIPs exploring different macro-architecture alternatives. We perform a schedulability analysis for each task clustering solution and we determine which one has the highest chances of meeting the deadline of the input applications and that should be considered in the next stages of the multi-ASIP design flow. A task clustering solution contains not only the information about the task clusters, but also the assignment of the data dependencies (that we identify as *messages*) to the communication architecture.

To address the circular dependency between the ASIP micro-architecture and WCET, we propose a WCET Uncertainty Model (UM), which captures the WCET of the tasks running on a range of possible ASIP micro-architecture configurations. We show how the UM can be inserted into the schedulability analysis for the evaluation of different task clustering.

We follow an approach similar to the Y-chart model described in Figure 1.5b: our schedulability analysis combines the application(s) and an initial platform model (in which the ASIPs have not been designed yet). The tasks within the application and the platform are combined through task clustering. Each task clustering solution is evaluated using the WCET estimations provided by the UM.

Our initial schedulability analysis is based on a task graph as application model. Additionally, we present an extension of our work in which the application is modeled as a SDFG. For each model, we apply a different static schedulability analysis and we demonstrate both analyses using SH tools for ASIP design. We also demonstrate how our DSE with UM, using both the task graph and SDFG models, can be integrated into a semi-automatic design flow (ASAM [7]) for the synthesis of a multi-ASIP platform.

Finally, we use our UM with a fixed priority preemptive scheduling (fpps), to demonstrate that it can be applied to different schedulability analyses, and we add a method for minimizing the area associated with a task clustering solution.

A list of publications and technical reports produced during the PhD program and that are included in this thesis is available on page vii.

1.7 Thesis outline

The remainder of this thesis is organized as follows. Chapter 2 describes the task graph application model, the platform model and the UM. We also present the experiments performed to define the UM.

Chapters 3 and 4 focus on the details of our schedulability analysis that is used to guide the macro-architecture DSE that defines the multi-ASIP platform. In Chapter 3, we explain how the UM can be included in a static schedulability analysis and how it can be used for comparing different task clustering solutions during DSE. We also describe the evolutionary algorithm used for DSE. In Chapter 4, we evaluate our DSE with UM for different case studies. In particular, we consider a Motion JPEG application. The results obtained have been verified implementing a hardware model of the multi-ASIP platform using SH tool. We also provide an analysis relative to the accuracy of the UM.

Chapters 5 and 6 present the extension of our schedulability analysis using a SDFG as application model. The SDFG allows exploiting both task level and pipeline parallelism in a more effective way than the task graph model. Chapter 5 focuses on the schedulability analysis that includes a Task-Level Analysis (TLA) and Pipeline Analysis (PA). In Chapter 6, we evaluate the schedulability analysis using multiple case studies, Motion JPEG, ECG and Spatial coding algorithm (extracted from MPEG4). Also for these case studies, the results obtained have been verified implementing a hardware model of the multi-ASIP platform using SH tool.

Chapter 7 describes how our DSE with UM is integrated within the Automatic Architecture Synthesis and Application Mapping (ASAM) design flow [7]. Both schedulability analyses, the one using a task graph and a SDFG are included into ASAM design flow. We also introduce an additional tool for macro-architecture DSE that is used in a later stage of the multi-ASIP design, when there is more accurate information about the micro-architecture configurations of each ASIPs. We also present the results obtained at each phase of ASAM flow for ECG case study.

In Chapter 8, we describe an additional extension of our DSE with UM. We apply a fpps policy and we consider also the area cost as optimization parameter during DSE. We show that the UM can be applied to different scheduling policy depending on the type of system that we are targeting. We use multiple case studies from E3S [2] benchmark to demonstrate the effectiveness of our approach.

Chapter 9 concludes this thesis and discusses issues that are open for future research.

1.8 Notes for the reader

In this thesis we use the following naming conventions. We call *task clustering* the assignment of the tasks to the different ASIPs (and of the messages to the communication architecture). The output of a *task clustering* is referred to as a *task clustering solution*. Each *task clustering solution* is composed of multiple groups of tasks. A *task clustering solution* also indicates how the *messages*, i.e., data dependencies, are associated with the communication architecture. Each group of task is called a *task cluster*. Each *task cluster* corresponds to an *ASIP*, which will be designed according to the tasks it has to run.

We use both terms *macro-architecture* or *system-level design* to refer to the platform design, i.e., the definition of the number and type of PE, their interconnections and the task clustering solution.

When we talk about *micro-architecture level*, we refer to the design of a single ASIP with its ISA and micro-architecture (i.e number of issue slots, RFs, MEM, etc.). We use the term *micro-architecture configuration* to indicate the micro-architecture resulting from a specific ASIP design.

The results obtained using SH (Intel) tools should not be used in any way as a reference to evaluate Intel technology or to compare SH tools with the tools of other competitors, as only a subset of the functionalities and optimization offered by the tools have been used and/or made available under our University license agreement.

CHAPTER 2

System models

This chapter introduces the different elements required to perform our macro-architecture DSE for multi-ASIP synthesis. As presented in Chapter 1, we are using a variation of a Y-Chart model (Figure 1.5b) for MPSoC design that requires two separate models to represent the input application and the hardware platform. Additionally, our approach for multi-ASIP synthesis requires the introduction of a *performance model*. In traditional MPSoC design the hardware components (processors and communication architecture) are already synthesized; hence, the performance of the application running on a given processor is modeled through its WCET or an average execution time (obtained through simulation or estimation). The problem that we are targeting has no synthesized processors (ASIP) available, so we introduce a model that captures the performance of the application running on a set of possible micro-architecture configurations of the ASIP. We called this model WCET Uncertainty Model (UM). This chapter is organized as follows: Sections 2.1 and 2.2 describe and justify the selection of the application and platform models, while Section 2.3 is dedicated to the description of the UM and the reasoning behind the model.

2.1 Application Model

The selection of the application model has a strong impact on the evaluation of a task clustering solution: distinct task clustering solutions can guide the ASIP and platform syntheses in different ways. We are mainly targeting real-time streaming applications (e.g., medical and multimedia applications as presented in Chapters 4 and 6). We also need a model capturing the intrinsic coarse-grain parallelism of the application to use it at macro-architecture level. We select a task graph that is widely used in embedded system design [38, 97, 86] and allows the partitioning of the application into multiple tasks. The tasks can run in parallel over multiple processors boosting the performance while preserving the data dependencies among them. Before providing a formal description of the task graph for application modeling, we introduce a general definition of Graph Model for parallel computation [87]:

DEFINITION 2.1 (GRAPH MODEL AS PROGRAM MODEL) A Graph Model can be used to represent the computation and the communication inside a program. The vertexes model the computation, while the edges model the communication. Each vertex is called a node and the computation associated with it *task*. A task contains sequential computation at different levels of granularity (atomic instruction/operation, a thread, a basic block or a sequence of these). A node is at any time involved in one activity, either communication or computation.

Starting from this general definition, we can define a task graph as follows [87].

DEFINITION 2.2 (TASK GRAPH) A task graph is a directed acyclic graph $G(V, E)$ that can be used as program model. V is a set of nodes that represent the tasks τ_j of the program, while E is the set of edges that model the communications between the tasks. An edge $e_{hj} \in E$ from task τ_h to τ_j (where $\tau_h, \tau_j \in V$), represents the communication from task τ_h to node τ_j. A non-negative weight, representing the communication cost, is associated with each edge.

We extend this definition of task graph to model our input application(s) (a similar model is described in [81]):

DEFINITION 2.3 (APPLICATION(S) MODEL) We have multiple applications A_i, each of them modeled as a task graph $A_i = G(V_i, E_i)$. Each application has a deadline d_i and a period T_i. Each task τ_j in A_i represents a unit of work, in our case, each task is a piece of sequential code that composes the application. We model the edges as *messages*, $m_g \in E$. The messages between tasks show the data dependencies and impose an execution order: with static scheduling, a task can start only after all its input messages have arrived. A value M_g is associated with each message and corresponds

to the amount of data it transmits expressed in bits (it represents the communication cost).

Figure 2.1 shows an example of an application model.

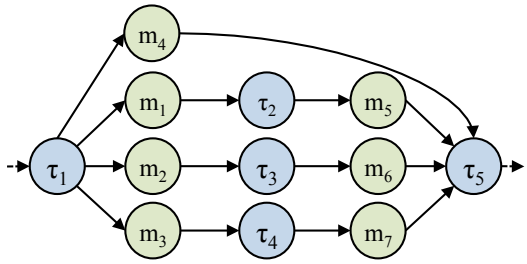


Figure 2.1: Example of an application model [69]

2.2 Platform Model

We define a generic platform model that can fit our requirements for DSE and evaluation of different task clustering solutions: the platform model should allow the fast estimation or simulation of performance of the entire system. We need a model in which we can easily distinguish between processing and communication elements to ease the association of tasks τ_j and messages m_g of the application(s) to the platform elements. Additionally, we need to capture the contention of the hardware resources.

In this thesis, we are considering heterogeneous bus-based multi-processor platforms that may contain multiple PEs as GPPs, ASICs, DSPs or ASIPs. In particular, we focus on platforms containing multiple ASIPs. The number and micro-architecture of the ASIPs are unknown and will be defined by our DSE. A platform instance contains a number of processors interconnected through a bus. However, we want to use a model that can be easily extended to consider other interconnection types as a link and a Network-on-Chip (NoC). In Section 6.6 we propose an example in which our DSE can be extended to a NoC.

Therefore, we model the platform as a graph called the Communication-Processing model (CP model).

DEFINITION 2.4 (CP MODEL) The communication-processing model (CP Model) is defined as a graph $CP = ((P, C), D)$, where P and C are finite set of vertexes, $P \cap C = \Phi$ and D is a finite set of edges. A vertex $PE_k \in P$ represents a processing

element (PE), while a vertex $CE_q \in C$ represents a communication element (CE). A directed edge $d_{ij} \in D$ represents a connection from the communication element CE_i (or PE_i) to the processing element PE_j (or CE_j).

All hardware components of the platform should be associated either with processing or communication elements. In our model we associate the processors (e.g. ASIPs, GPPs, etc.), HW accelerators, switches, and external memories with PEs, while we associate links and buses with CEs. We associate all hardware components with processing capabilities, or functionalities different from the communication, with the P set. Switches are considered as PEs, as they can contain more complex functionalities than a bus (e.g. they can take routing decisions and analyze the network congestion).
The CP model is a bipartite graph, i.e., processing elements are always interconnected through communication elements. This allows a uniform model that can be easily verified. Additionally, it allows modeling a platform with different levels of granularity: for example, the same platform can be represented splitting a processing element into multiple processing and communication elements to capture all the relevant hardware components for the designer. An example of a bus-based platform and its corresponding CP model are shown in Figures 2.2a and 2.2b: we have three ASIPs and a subsystem that are modeled as PEs, while the bus is modeled with a CE. In Figures 2.2c and 2.2d, there is an example of sub-system, with an ASIP, a memory and a hardware accelerator, and its corresponding CP model. The CEs in the subsystem are modeling point-to-point links. In this thesis, for the figures representing CP models, we use the following convention to simplify the graphic representation. An undirected edge corresponds to a full-duplex connection (two directed opposite edges).
A processing or communication element may have some characterization parameters that allow the estimation of the performance of the entire system when implementing a specific task clustering solution (also other parameters as area or power can be included if required by the evaluation).
In this thesis, we are considering a bus-based platform with multiple ASIPs, where the k-th processing element, PE_k, has a frequency f_{PE_k}. As mentioned in Chapter 1, we consider ASIPs that have not been implemented yet; therefore, we cannot know the WCET $C_j^{PE_k}$ of a task τ_j running on a specific ASIP PE_k. We model the performance of each task with our UM presented in Section 2.3. For the other types of processors, e.g., GPP, we suppose to know the WCET $C_j^{PE_k}$.
A bus is characterized by a frequency and a bandwidth. We denote by $b_w^{f_b}$ a bus with a bandwidth w and a frequency f_b. Recalling that M_g is the size of the message in bits, we calculate the transmission time for a message m_g on the bus $b_w^{f_b}$ as $C_{m_g} = \frac{M_g}{w * f_b}$. The processing elements and the bus can have different frequencies.

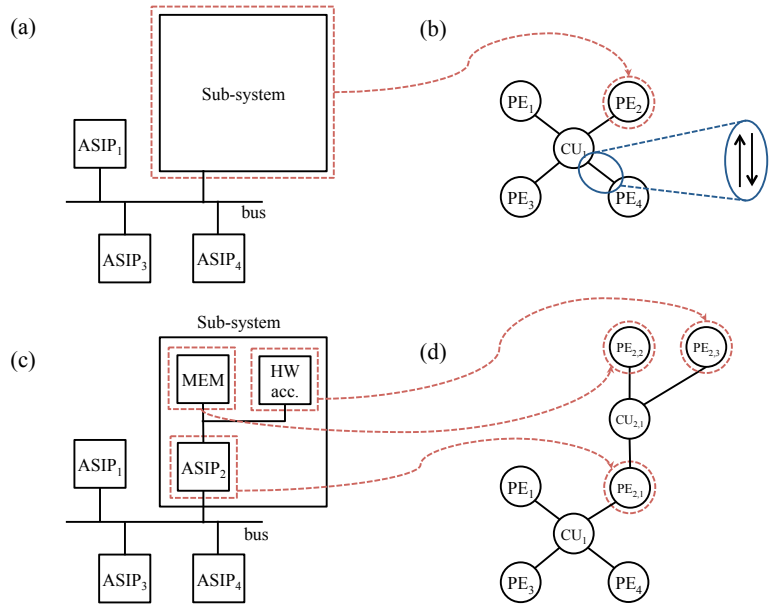


Figure 2.2: Examples of platform (a,c) and its corresponding CP models (b,d)

2.3 Modeling WCET uncertainties

In this section, we present our UM and how we use it to model the performance of a task τ_j assigned to an ASIP, PE_k. Tasks are grouped into clusters: a task cluster corresponds to a processing element PE_k. We recall that a task clustering solution includes multiple task clusters and the assignment of the messages to the communication architecture. As mentioned in Chapter 1, the WCET value depends on the ASIP micro-architecture, which is implemented depending on how tasks are clustered: the WCET of each task can be known only when the ASIP design is available and, therefore, its micro-architecture is defined. However, the ASIP micro-architecture can be established only after we decide the tasks that are assigned to a specific ASIP. This creates a circular dependency between the need of knowing the implementation of the ASIPs for estimating the global system performance and the impossibility of determining them in the early design phases when the task clustering is still unknown. This circular dependency is the reason for the introduction of an uncertainty model that captures the WCET of the entire spectrum of possible ASIP micro-architecture, for a task τ_j: the WCET of each τ_j is modeled as a stochastic variable C_j and the associated probability distribution function. Such uncertainty models are used in practice in the early design stages [8].

Note that the variability of the *worst-case* execution time C_j of a task τ_j is due to the variation among the possible ASIP implementations on which task τ_j will run, and does not reflect the variation in execution time, which is due to variation in the input data and modern architectural features, e.g., pipeline and branch prediction. The final implementation of the ASIP running τ_j will only be available after the time-consuming ASIP micro-architecture design (as presented in Section 1.1.2). We use the probability distribution of C_j during DSE as it is unfeasible to design every ASIP micro-architecture resulting from a change in the task clustering.

We assume that the designer captures the probability distribution function of the WCET C_j of a task τ_j using two bounds: the smallest WCET value C_j^l (lower bound) and the largest value C_j^u (upper bound). The designer can arrive at these two values based on his or her knowledge of the functionality of the task and the possible range of ASIP micro-architectures. These values can also be estimated; the lower WCET bound can correspond to the expected WCET when τ_j is executed on an ideal processor according to an as soon as possible (ASAP) scheduling without architectural constraints. Instead, the upper WCET bound can correspond to a sequential execution of τ_j on the slowest ASIP as possible. For the case studies in Chapters 4 and 6, we use an external code analysis tool [43] to get the upper and lower bounds values; additionally, in the same chapters, we discuss the sensitivity of our approach to the accuracy of the WCET bounds. Within these two values, we use a normal distribution for C_j that models the WCETs of the task executing on an undefined ASIP that has not been designed yet. The reasons for using a normal distribution are provided in Section 2.3.1. A normal distribution is defined by its probability density function (PDF) (Equation 2.1), where μ_j and σ_j^2 are the parameters defining the distribution of C_j. μ_j is the mean or expectation and σ_j^2 is the variance (σ_j is the standard deviation).

$$f_j(x) = \frac{1}{\sigma_j\sqrt{2\pi}} e^{-\frac{(x-\mu_j)^2}{2\sigma_j^2}} \tag{2.1}$$

In our UM, we use the definition of cumulative distribution function (CDF), $F_j(x)$, of a normal distribution (Equation 2.2).

$$F_j(x) = \frac{1}{2}\left[1 + erf\left(\frac{(x-\mu_j)}{\sigma_j\sqrt{2}}\right)\right] \tag{2.2}$$

The CDF, $F_j(x)$ of C_j, can be specified as $F_j(x) = P(C_j \leq x) = p$, where the probability p is an indicator of the number of ASIP configurations that lead to a C_j smaller than a value x. Figure 2.3 shows an example of CDF for a task τ_j. The distribution is built such that $P(C_j \leq C_j^u) \approx 1$. This means that task τ_j will finish in C_j^u time units or less on *all* possible ASIP micro-architecture configurations. At the same time, we also assume that $P(C_j \leq C_j^l) \approx 0$[1]. This means that according to the designer's

[1]The CDF of the normal distribution does not reach the one and zero values, therefore we use values that approximate them.

evaluation, *none* of the possible micro-architecture configurations will finish faster than C_j^l time units. Using the C_j^u and C_j^l values, we calculate μ and σ^2 characterizing the normal distribution of C_j. μ is calculated as the mean value between C_j^u and C_j^l (Equation 2.3). σ^2 is calculated using Equation 2.5. Equation 2.5 derives from the quantile function $F^{-1}(p)$ defined by Equation 2.4. The quantile function is the inverse of the CDF. Once we have μ and σ^2, we can draw the corresponding CDF.

$$\mu = \frac{(C_j^u + C_j^l)}{2} \tag{2.3}$$

$$F^{-1}(p) = \mu + \sigma\sqrt{2}erf^{-1}(2p-1) \tag{2.4}$$

$$\sigma = \frac{F^{-1}(p) - \mu}{\sqrt{2}erf^{-1}(2p-1)} \tag{2.5}$$

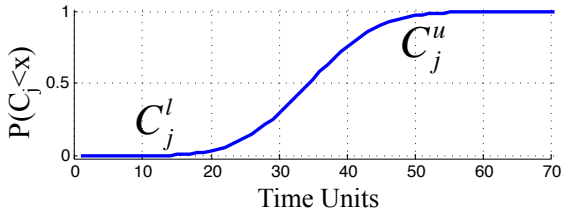


Figure 2.3: Example of UM for a task τ_j [69]

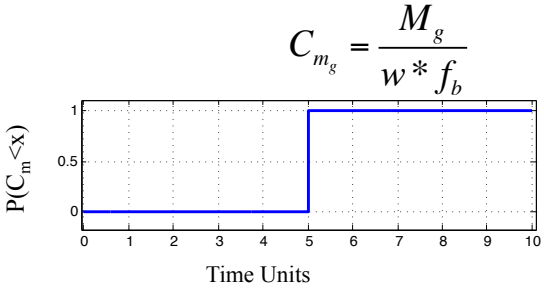


Figure 2.4: Example of C_{m_g} for a task m_g [69]

We also build a model for the messages to use during DSE. For each message m_g we know its size in bits. Additionally we know the frequency and data width of the bus that we want to explore. Given these elements, we can calculate the worst-case transmission time C_{m_g} of each message m_g as presented in Section 2.2. C_{m_g} is a single value and *not* a stochastic variable: for each type of bus that we want to explore,

we have a different C_{m_g}. It is possible to represent the C_{m_g} of a message as a step function 2.6 (example in Figure 2.4).

$$u(x - C_{m_g}) = \begin{cases} 0 & \text{if } x < C_{m_g} \\ 1 & \text{if } x > C_{m_g} \end{cases} \tag{2.6}$$

If during DSE, we want to consider already synthesized ASIPs or intellectual property (IP) cores (e.g. GPPs or DSPs), we need the WCET of the tasks running on these components. As for the messages, we can model the WCETs of tasks executing on these processors as step functions. If the WCET of a task τ_j on any of these PEs, PE_k, is $C_j^{PE_k}$, then the CDF of τ_j can be expressed as $F(x)_j = P(C_j = C_j^{PE_k}) = 1$. In Chapter 3 we explain how to combine the probability distribution of the tasks assigned to ASIPs with the WCET of messages (or of task assigned to IP cores) to perform a schedulability analysis and compare different clustering solutions.

2.3.1 Validation of the normal distribution for UM

In this section, we explain the motivation for selecting a normal distribution for the modeling of the stochastic variable C_j associated with a task τ_j. To determine the distribution type, we performed two different types of experiments:

- In experiment 1 we consider a single task τ_j and, using an ASIP VLIW simulator, we prove that the normal distribution is the best one for approximating the WCETs of τ_j running on a considerable number (~ 500) of ASIP micro-architecture configurations.
- In experiment 2 we consider an application A_i, modeled as a task graph and we assume different probability distributions (normal, Gumbel and uniform) for modeling the WCET of the tasks of A_i; then we run our DSE for finding the best task clustering solution. We determine that the normal distribution is the best choice for guiding our DSE.

In the next subsections, we present the details of experiments 1 and 2.

2.3.1.1 Experiment 1

In this experiment we use a single task τ_j and we verify the distribution of the WCETs obtained from running the task on multiple micro-architecture configurations of a VLIW

ASIP. We want to determine how the WCET of the task varies depending on the micro-architecture configurations, and prove that our WCET uncertainty model proposed in Section 2.3 is able to capture this variation.

We performed this experiment on two **tasks**, τ_{jpeg} and τ_{mp3} of different sizes and complexities. τ_{jpeg} contains the sequential code of a JPEG decoder [94], while τ_{mp3} contains the sequential code of a MP3 decoder, part of the MAD library [3]. We considered one task at a time and we ran it on a VLIW architecture that can be configured similarly to the ASIP architectures considered in this thesis. We used the VLIW Example (VEX) [26], which is a VLIW C compiler and simulator developed at Hewlett Packard Research Laboratories. The micro-architecture of VEX is highly and easily configurable using a micro-architecture configuration file. In Figure 2.5 there is the default micro-architecture configuration of VEX that provides an overview of the micro-architectural components available in VEX. In Table 2.1 there are the micro-architecture features that we explored and used to captures the variability in the WCET of the micro-architecture designs. Thus, we varied the number of arithmetic and logic units (ALU), the multipliers (MUL), the registers in the register file (RF) and the issue, load and store slots. We set the design space to these parameters considering the features of VLIW processors available on the market (e.g. [5]) and the characteristics of the tasks τ_{jpeg} and τ_{mp3}. For each micro-architecture configuration, VEX performs the compilation of the C code of the task, simulates its execution and returns the number of execution cycles.

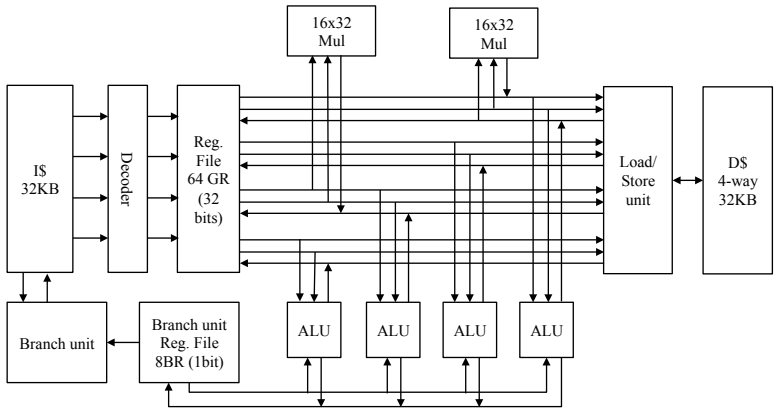


Figure 2.5: Default VEX micro-architecture configuration [25]

Using the parameters in Table 2.1 we evaluated a large number of micro-architecture configurations; due to some limitations of VEX, some combinations of parameters are not accepted. In total, we simulated 490 micro-architecture configurations for the MP3 decoder task and 560 for the JPEG decoder task.

Table 2.1: Micro-architecture features explored

Task	*Issue width*	*num. ALU*	*num. MUL*	*RF size*	*Load slot*	*Store slot*
MP3 decoder	1,2,3, 4,5,6, 7,8	4,5,6, 7,8	2,3,4, 5,6,7, 8	32, 64	4	2
JPEG decoder	2,3, 4,5,6, 7,8	4,5,6, 7,8	1,2,3, 4,5,6, 7,8	32, 64	4	2

For each micro-architecture configuration, we compiled and ran the task. We used the m3explorer tool [102] for performing DSE in an automatic way; m3explorer is a generic tool for DSE that can be interfaced to any simulation/evaluation tool using XML files. We used the tool to perform an exhaustive DSE. Using scripting languages, we created the interfaces between VEX and m3explorer: the scripts automatically generated the micro-architecture configuration file and collected the results (number of cycles) produced by VEX. For each micro-architecture configuration explored, we assumed a frequency of 100 MHz to calculate the execution time in ms (given the characteristics of the micro-architectures explored, we can safely assume a frequency of 100 MHz by comparison with other commercial VLIW processors, e.g. [5]). For a particular micro-architecture, after simulations with multiple input files, we considered as WCET the largest value of the execution time. We know that such a value does not represent the WCET, which is a theoretical upper bound determined through analysis, but we believe this value is a good approximation for our experiments.

The results for the MP3 decoder are presented in Figures 2.6a and 2.6b and those for JPEG in Figures 2.7a and 2.7b. Figures 2.6a and 2.7a present with a bar graph the distribution of the WCET of τ_{jpeg} and τ_{mp3} obtained with our experiments. We used a fitting function of MATLAB to determine the probability distribution type that better approximates these WCET values; we wanted a distribution that is a good approximation of the WCET of both tasks τ_{jpeg} and τ_{mp3}.

We observed that a normal distribution is a reasonable approximation (represented with a continuous red line in Figure 2.6a and 2.7a). The corresponding CDF are plotted in Figure 2.6b and 2.7b. Each figure shows two CDF curves: the CDF resulted after experiments (depicted with a continuous blue line) and the CDF obtained by using our model (the green dotted line). Our WCET model (the green dotted CDF) was obtained as explained in Section 2.3, considering a normal distribution between a lower bound C^l and an upper bound C^u of the WCET (we took the fastest and lowest micro-architecture configurations).

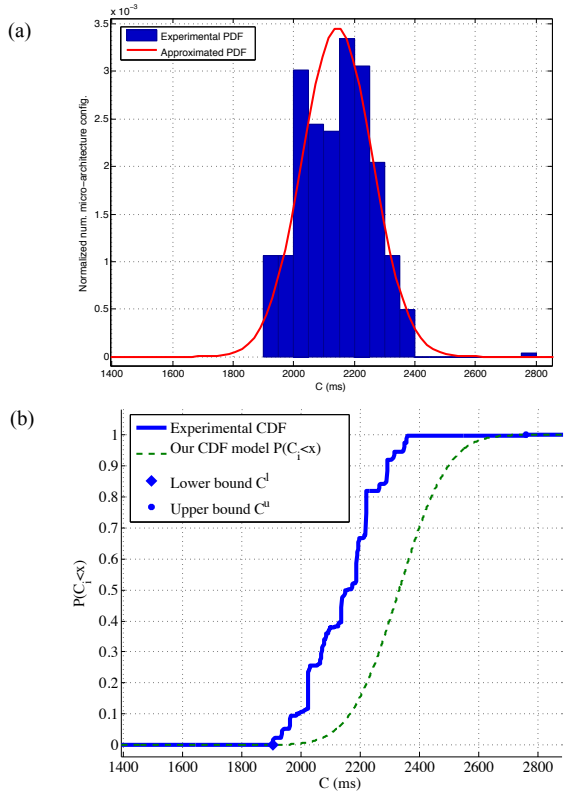


Figure 2.6: (a) Histogram of the probability density function of the C_i obtained with VEX and (b) comparison of our proposed CDF ($P(C_i < x)$) with the simulation results for mp3 decoder task

The micro-architectures, corresponding to the upper and lower bounds of the WCET for the two tasks, are summarized in Table 2.2. The slowest and fastest micro-architecture configurations do not correspond to the smallest (sequential) and biggest (most parallel) one. We presume that this is due to the scheduling done by VEX compiler. For example let us consider two identical micro-architectures that differ only in the issue width number. The issue width determines the total number of operations in a VLIW instruction and therefore, the operation-level parallelism. We would expect better performance from the micro-architecture with a bigger issue width as potentially can run more operations in parallel; or at least we would expect similar performances from the two configurations, in case the code of the task does not allow operation level parallelism. Instead, in our experiments, we verified that a more parallel micro-architecture configuration can lead to worst performance than a smaller one and this is probably due to a wider number of scheduling options for the compiler as there are more resources

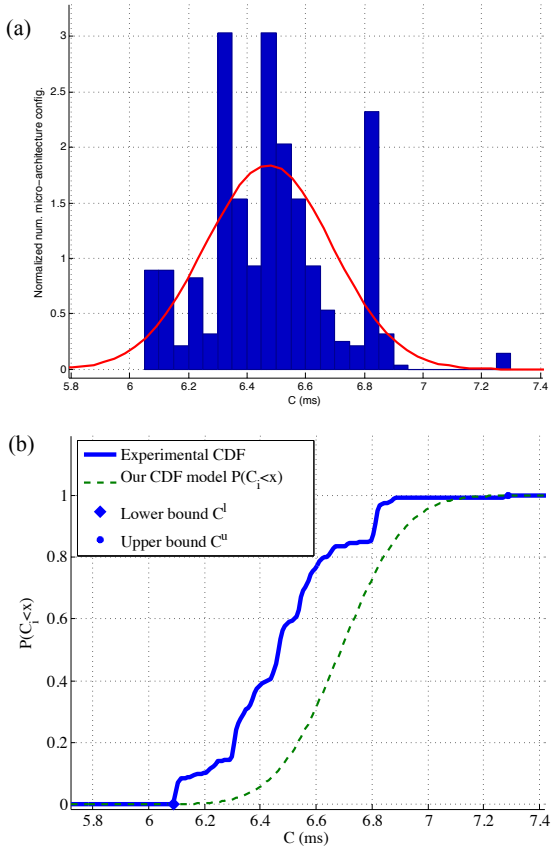


Figure 2.7: (a) Histogram of the probability density function of the C_i obtained with VEX and (b) comparison of our proposed CDF ($P(C_i < x)$) with the simulation results for jpeg decoder task

that can be used. For example, the lower bounds of τ_{mp3} and τ_{jpeg} have been found using a smaller issue width (5 and 7, respectively) than the maximum allowed by our design space (8).

This experiment showed that the WCET has a normal distribution and that our proposed uncertainty model is a valid and safe approximation. Note that the CDF of our model leads to more pessimistic (larger) WCETs compared to experimental measurements. This is acceptable as we want to consider the worst case of the execution time and the WCETs produced by our experiments might be optimistic (smaller), since they are not a theoretical upper bound obtained through analysis. It is important to mention that the proposed WCET uncertainty model is used only for design space exploration, and not

for providing timing guarantees.

Table 2.2: Micro-architectures associated with the WCET upper and lower bounds

Task	*WCET*	*Issue width*	*num. ALU*	*num. MUL*	*RF size*	*Load slot*	*Store slot*
MP3 decoder	C_l	5	8	5	64	4	2
	C_u	1	4	2	64	4	2
JPEG decoder	C_l	7	5	3	64	4	2
	C_u	2	8	6	64	4	2

2.3.1.2 Experiment 2

The purpose of the second experiment is to validate the results from Experiment 1. In Experiment 1, we consider isolated tasks running on a single ASIP and we calculate the WCET distribution of the same task running on multiple micro-architecture configurations; on the other hand, in this experiment, we consider an entire application modeled as a task graph and we run our DSE with UM to determine the best task clustering solution. The WCET of each task in the application is modeled using a different probability distribution: normal (N), Gumbel (G) and uniform (U) distributions. We want to verify if a performance model based on the normal distribution is able to properly guide the DSE and to find the best clustering solution when compared to other probability distributions. We select the Gumbel distribution as it is commonly used to model the WCET [22, 36] and the uniform distribution for its simplicity. The details of the DSE and of the schedulability analysis performed are presented in Chapter 3; for the sake of this experiment, it is enough to know that during DSE, we can compare different task clustering solutions performing a schedulability analysis. The schedulability analysis combines the WCET probability distributions of the tasks; its output can be used to discriminate among the different task clusterings and identify the best one.

We performed the same DSE using the normal, Gumbel and uniform distributions. For each task τ_j of the input application A_i, we have the upper and lower bound, C_j^l and C_j^u, and we calculated the CDFs of each task according to the three types of distribution. In Figure 2.8 there is an example of the probability density function and of the corresponding CDF for normal, Gumbel and uniform distributions.

Additionally, we ran the same DSE and schedulability analysis, but with a *deterministic* WCET for each task (i.e. we know the WCET value and there is no probability distribution associated with it). Using deterministic WCET, we can calculate the exact scheduling length.

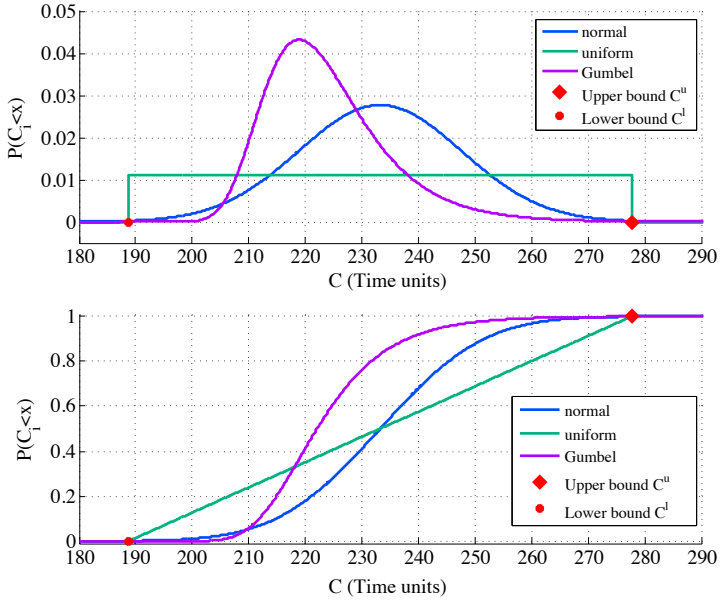


Figure 2.8: (a) Probability density function and (b) Cumulative distribution function (CDF)

We then compared the task clustering solutions obtained using the three types of distributions to the one obtained using deterministic WCETs. In this section we use the following terminology: DSE_{UM}^{type} indicates our DSE with UM and $type$ is the distribution used, $type = \{N, G, U\}$, while DSE_{det} indicates the DSE using deterministic WCETs. As we want to compare the results obtained with DSE_{UM}^{type} and DSE_{det}, we need to guarantee that the two DSE are comparable. Therefore, we need to use the same optimization function during DSE. In the deterministic DSE the optimization function is the minimization of the scheduling length. Instead, in our default implementation of the uncertainty model (described in Chapter 3) the optimization function is the maximization of the probability of meeting the application deadline. We modified our DSE_{UM}^{type}, so that the exploration is guided by the minimization of the scheduling length at different probabilities. We used the inverse of the CDF, i.e. the quantile function $C_i = F^{-1}(p)$, to obtain the C_i of a clustering solution at specific probabilities p_i, where $p_i \in P_i = \{0.02, 0.50, 0.98\}$. We selected three different probabilities to take the shape of the different CDFs into account and not to favor any distribution types.

For each probability and for each distribution type we ran the DSE_{UM}^{type} obtaining different optimized clustering solutions. In total, we ran the DSE_{UM}^{type} nine times, one for each combination of p_i and distribution type; we also obtained nine different clustering

solutions. These solutions are the results of an evaluation that takes into account the entire range of possible micro-architecture configurations of the ASIPs.

Then we ran the DSE_{det} multiple times: for each execution of the DSE_{det} and for each task τ_j in our input application A_i, we assigned a deterministic WCET randomly extracted from the range $[C_j^l, C_j^u]$. We assume that this WCET corresponds to a specific ASIP micro-architecture configuration. Using these deterministic WCET values, we performed the DSE_{det} to find the best clustering solution that minimize the scheduling length when specific ASIP micro-architecture are used. We ran the DSE_{det} 5,000 times, each time randomly extracting, for each task τ_j, a WCET from the range $[C_j^l, C_j^u]$. Therefore we obtained 5,000 *sets* containing a WCET value for each task in the application.

Once we collected all the results from the nine executions of the DSE_{UM}^{type} and of the 5,000 executions of the DSE_{det}, we compared them. The schedule length obtained with DSE_{det} represents the optimal scheduling that we can obtain when knowing the exact values of the WCET for each task. We took the nine clustering solutions produced by the DSE_{UM}^{type} and we calculated the real scheduling length of each of them using, for each task, instead of the WCET probability distribution, the deterministic WCET values also used for the DSE_{det}. This means that we calculate 5,000 scheduling length for each of the nine task clustering solutions.

Then we compared the scheduling length of the 5,000 clustering solutions found through DSE_{det} with the scheduling length of the clustering solutions found with DSE_{UM}^{type}, which are evaluated with the same WCET values used for the deterministic method. Our objective is to identify which probability distribution type allows finding a task clustering solutions with the closest scheduling length to the one found with DSE_{det} in which we have precise WCET for the tasks.

We ran this evaluation on six synthetic case studies, which characteristics are specified in Table 2.3. In Table 2.4, for each case study, and for each distribution type (at $p_i \in P_i = \{0.02, 0.50, 0.98\}$) we have the *average relative error* in the scheduling length obtained with the UM when compared to the DSE_{det}. The *average relative error* is calculated as follows. Let us consider the task clustering obtained using a normal distribution (DSE_{UM}^{N}) and the quantile function at $p_i = 0.50$. We evaluated this task clustering solution with the 5,000 *sets* of WCET (the same used during DSE_{det}). We compared one by one the scheduling length for each of the 5,000 *sets* and we got a relative error for each one of them. Then we calculated the average of the errors and we got the *average relative error* that we used to fill in Table 2.4.

We observed that the uniform distribution is the one with the worst approximation. Even if normal and Gumbel distributions return comparable errors for some case studies, the first one is a better fit for most of the case studies.

Table 2.3: Case studies for the comparison of CDF types

Case Study ID	1	2	3	4	5	6
No. of Apps.	4	5	18	10	39	48
No. of Tasks	14	15	24	26	44	60
No of ASIPs	11	14	14	11	10	13

Table 2.4: Results of the comparison of CDF types (% average relative error)

	normal			Gumbel			uniform		
Case Study ID	**0.02**	**0.50**	**0.98**	**0.02**	**0.50**	**0.98**	**0.02**	**0.50**	**0.98**
1	0.02	0.02	0.02	0.02	0.02	0.02	0.04	0.05	0.04
2	0.06	0.13	0.06	0.11	0.04	0.07	0.11	0.11	0.11
3	0.16	0.05	0.05	0.18	0.05	0.05	0.08	0.16	0.98
4	0.05	0.06	0.06	0.08	0.22	0.05	0.07	0.06	0.15
5	0.07	0.09	0.11	0.10	0.09	0.13	0.12	0.13	0.12
6	0.06	0.05	0.06	0.05	0.06	0.05	0.06	0.06	0.08

Figure 2.9 and 2.10 represent the distributions of the relative error, i.e. percentage difference in the scheduling length for case study 1 and 4 (Table 2.3). The distribution of the relative error for the other case studies are available in Appendix A.1. For each case study, the normal, Gumbel and uniform distributions are represented. For each distribution, we grouped together the results obtained for $p_i \in P_i = \{0.02, 0.50, 0.98\}$, for a total of 15,000 evaluated schedulings. The normal distribution is the one that returns schedulings with length closer to the DSE_{det}. In fact for all case studies is the one with the highest number of scheduling length difference equal zero. For example, let us observe Figure 2.9. Each histogram contains the results for 15,000 schedulings (5,000 schedulings for each p_i). The histogram containing the results obtained with a normal distribution, has most of the schedulings ($\sim 8,000$) with 0% error when compared to the schedulings obtained with DSE_{det}. This value decreases to ~ 3000 for Gumbel and ~ 1800 for the uniform distributions. Depending on the case studies, the distribution of the errors can vary: we observed that for the case studies with a higher number of tasks, the errors are centered around zero, but are distributed on a wider range. However, in all cases, we verified that the normal distribution produces scheduling length closer to the one obtained using a deterministic DSE, in which we know the precise value of the WCETs of each task.

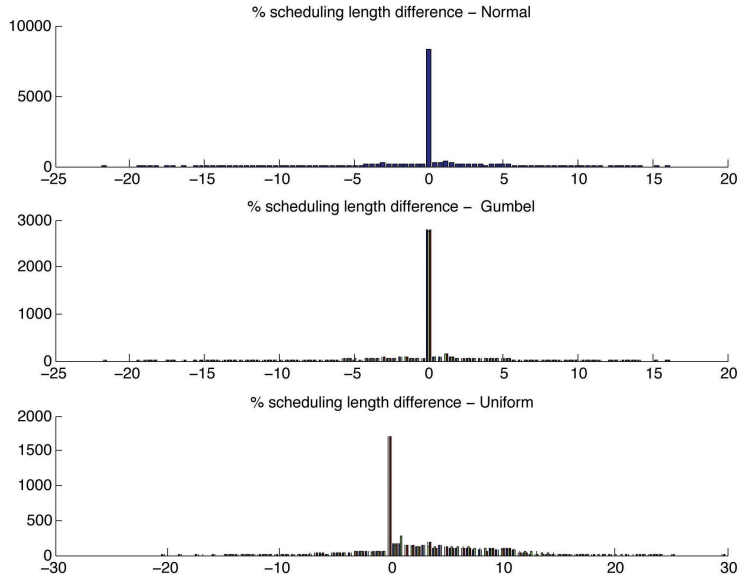


Figure 2.9: Histogram of the percentage (%) differences in the scheduling length for Case Study 1

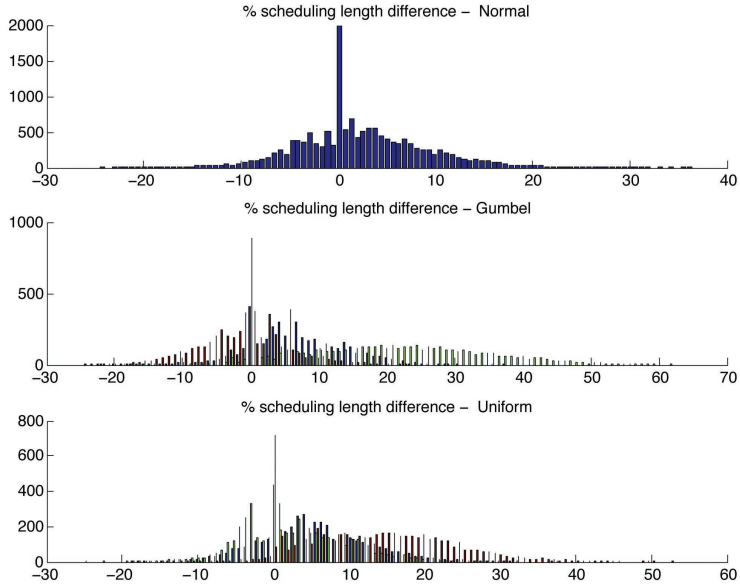


Figure 2.10: Histogram of the percentage (%) differences in the scheduling length for Case Study 4

2.4 Summary

In this section we introduced the task graph model that we use to represents our input application A_i and the CP model that we use for the platform representation.

Additionally, we presented our UM, i.e. the performance model that we use for modeling the WCET of each task τ_j.

The WCET of each task depends on the ASIP micro-architecture, which is designed depending on how the tasks are clustered. We use a probabilistic performance model to break the circular dependency between the need of knowing the implementation of the ASIPs for estimating the global system performance and the impossibility of knowing the ASIP micro-architectures in the early design phases when the task clustering is still unknown. Our UM captures the entire spectrum of possible ASIP micro-architectures, for each task τ_j: the WCET of each τ_j is modeled as a stochastic variable C_j and the associated probability distribution function.

Moreover, we described the experiments that we performed to justify the selection of a normal probability distribution to model the stochastic WCET.

In the next session we are going to explain how to use the application, platform and UM models to perform our schedulability analysis and identify the best task clustering solutions.

CHAPTER 3

Macro-architecture level DSE

In this chapter we describe our macro-architecture DSE and the schedulability analysis that we perform to evaluate different task clustering solutions. Our schedulability analysis is implemented using the application and platform models and the UM described in Chapter 2. In Section 3.1, we use a motivational example to give an overview of our schedulability analysis to the reader and to present how it can be used to discriminate between different task clustering solutions. Then, in Section 3.2, we present the problem formulation, and finally, in Section 3.3, we describe our schedulability analysis for the evaluation of a single task clustering solution and the DSE algorithm for exploring the solution design space. Part of the material described in this chapter has been published in [69].

3.1 Motivational Example

The schedulability analysis of a task clustering for a given application A_i is done according to the application deadline d_i. We have an initial platform, specified as a CP model. We consider a multi-ASIP bus-based platform, in which the ASIPs have not been designed yet: the CP model indicates the maximum number of PEs that should be included into the final system (our platform cost, PC_{max}) and how they are interconnected. Additionally, we have the UM of each task, τ_j in A_i, and the WCET, C_{m_g}, for each message (according to the bus type).

Let us consider the application task graph in Figure 3.1a and the UM of each task in Figure 3.1c. The initial platform model is in Figure 3.1b. We evaluate two different task clustering solutions Sol_1 and Sol_2 (Figure 3.2a); our schedulability analysis produces the CDFs for the task clustering solutions (Figure 3.2b). We obtain them combining the CDF of each task and the WCET of the messages. Each solution is evaluated according to the application deadline (d) and we obtain a probability p_{sol}. Sol_1 has a probability $p_1 = 0.82$ while Sol_2 has $p_2 = 0.03$. This indicates that the first task clustering solution is better than the second one: a higher probability indicates that a task clustering is more likely to meet the application deadline when the platform is designed.

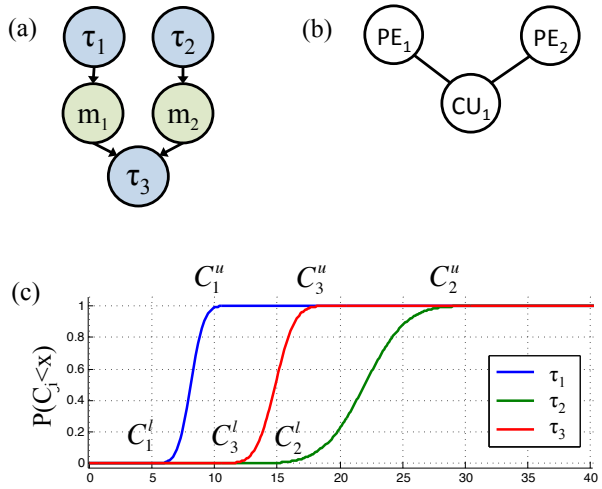


Figure 3.1: Example of schedulability analysis: (a) input application, (b) PC model of the input platform and (c) UM of the tasks [69]

3.2 Problem Formulation

Given one or multiple applications A_i (see Section 2.1) with deadline d_i, and a platform cost constraint PC_{max}, the problem is to define a system-level multi-ASIP platform, such that the probability of having a schedulable implementation is maximized under the specified cost constraint PC_{max}.

Defining a system-level platform means performing a DSE to decide the task clustering and the interconnection. Our UM takes as input the task graphs of the applications, their deadlines and the cost constraint PC_{max} that is defined as the maximum number[1]

[1]We will consider the ASIP area cost in Chapter 8.

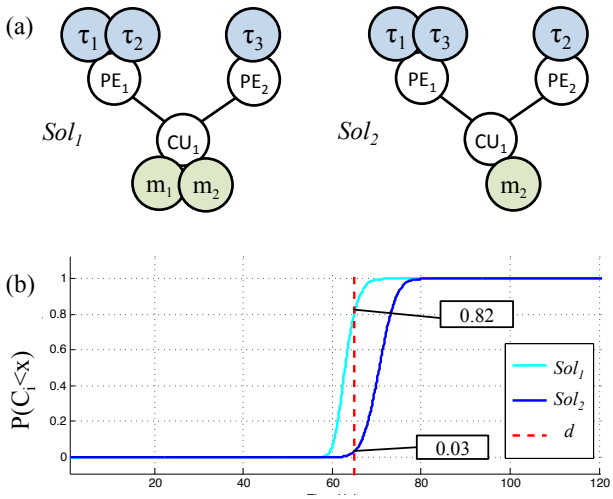


Figure 3.2: (a) Task clustering solutions and (b) corresponding CDFs produced by the schedulability analysis of the example in Figure 3.1 [69]

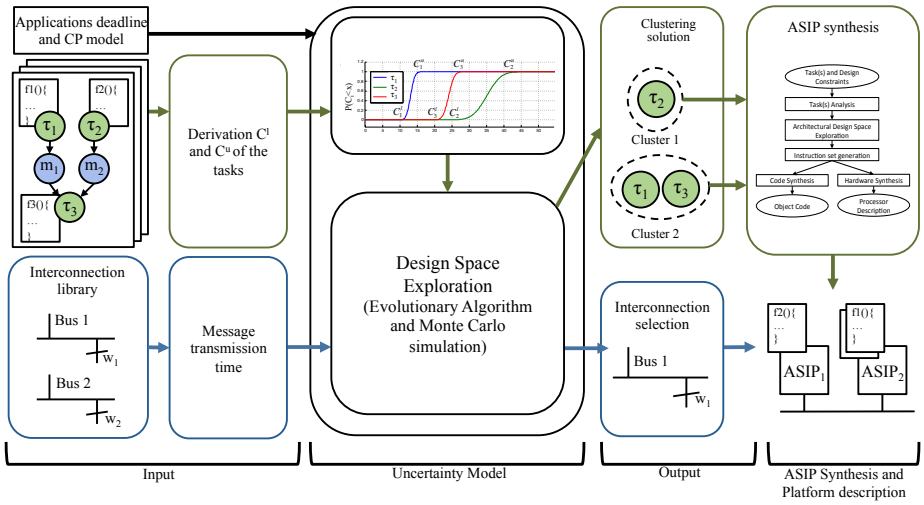


Figure 3.3: Multi-ASIP platform design flow [69]

of ASIPs that can be included in the platform. Moreover, we consider a library of buses with different speeds and bandwidths, from where the DSE selects the appropriate bus. There can be a set of legacy components that have to be used in the architecture and some tasks may be assigned to some specific PEs by the designer. Our optimization takes these constraints into account. The designer, based on his knowledge or through an analysis of the task code, can provide the upper and lower bounds for the WCET or can use a profiling tool as done for our experimental evaluation (Section 4.2). Given the size in bits of each message and the library of buses, it is possible to estimate the communication time for each message. The DSE evaluates different task clustering solutions as presented in Section 3.3.1 and selects the ones which maximize the probability of having a schedulable implementation. After DSE, we use an ASIP design flow (Figure 1.2) to implement an ASIP for each task cluster. The output of our multi-ASIP platform design approach is a platform architecture, consisting of several ASIPs and possibly also legacy components, and their interconnection (a selected bus with a certain speed and bandwidth). The design is performed under the platform cost constraint PC_{max}. Without any loss of generality, our approach can be used also if the initial CP model of the platform includes a set of legacy components (e.g. ASICs, DSPs). Figure 3.3 shows our flow for the design of a multi-ASIP platform.

3.3 Platform Definition using an Evolutionary Approach

In this section, we present the details of the schedulability analysis with UM and the DSE that we use to explore the design space of task clustering solutions. The design space can be huge and cannot be exhaustively explored, so an Evolutionary Algorithm (EA) has been implemented. The objective function guiding the exploration is the maximization of the probability p_i (returned by our schedulability analysis) for a certain clustering solution to meet the deadline, under a platform cost constraint PC_{max}. Section 3.3.1 illustrates how a single task clustering solution is evaluated, Section 3.3.2 explains the policy to compare two task clustering solutions, Section 3.3.3 presents the EA used for the DSE and finally Section 3.3.4 describes the application of our UM through a small example.

3.3.1 Schedulability Analysis

Our approach is meant to be used at design time, when there is no platform available. For the final implementation of our platform we use single-threaded VLIW processors (provided by SH). Therefore, for our schedulability analysis, we use a *static scheduling policy*. We also assume a non-preemptive scheduling.

In a classic scheduling problem, when the WCET of each task is known, it is possible to determine the schedule table and thus to perform a schedulability check. This cannot be done in our case, as the WCET is expressed by a stochastic variable. Instead, we perform a schedulability analysis of each task clustering solution and use this as the basis for calculating the objective function during DSE. Note that the analysis presented in this section is only used to guide the search, not to provide schedulability guarantees. We assume that a detailed schedule table will be built during the later design and development stages, when more accurate information about WCETs and the ASIP micro-architecture is available.

With static cyclic scheduling, δ_{A_i} is the length of the schedule for A_i. As previously mentioned, we cannot calculate the schedule length of a task clustering solution, but we can perform a schedulability analysis to determine the probability of having a schedule table that meets the deadline, as δ_{A_i} is a stochastic variable as well. Thus, the probability of A_i to meet the deadline d_i is defined as $P(\delta_{A_i} \leq d_i)$. In [61], the authors present how to determine δ_{A_i} in case of stochastic execution times (WCETs in our case), using an analytical approach that relies on the assumption of independence between the starting and finishing time of each task/message. This assumption in our case does not hold, so we substitute the analytical method with Monte Carlo simulation (MCS), which, in addition, is able to take into account the task dependencies and provide more precise results.

With MCS, we repeat the schedulability analysis of a task clustering solution multiple times. At each iteration of MCS, we extract a WCET value for each task according to its probability distribution. The output of a *single* iteration of the MCS is the schedule table when considering the extracted WCETs. We collect the output of *all* iterations of MCS and we obtain the probability distribution of δ_{A_i}. Finally, we calculate the probability $p_i = P(\delta_{A_i} \leq d_i)$ of an application A_i to be schedulable. The steps for performing MCS are summarized in Figure 3.4a.

As a *MCS* is known for being time consuming, a reduced number of samples, n has been used to speed up the execution. The results obtained with $n = 5,000$ samples have then been compared to results obtained with $n = 50,000$ samples, and the difference in the value of p_i between them was less than 3%.

We use an approach similar to As Soon As Possible (ASAP) [96] scheduling to calculate δ_{A_i}. Let us illustrate how δ_{A_i} is obtained using the application from Figure 3.6a and the platform implementation solution from Figure 3.6b. The clustering solution in Figure 3.6b uses two ASIPs, PE_1 and PE_2. Messages are assigned to the bus (CE). We start by identifying a layer subdivision [61] of the task graph, see Figure 3.6c. If communicating tasks are assigned to the same PE, the communication cost is ignored. Each task and message has a priority assigned (defined by the task id, to a lower id corresponds an higher priority). A layer identifies the tasks and messages of the applications A_i that can be executed in parallel, i.e. that have no data dependency and are

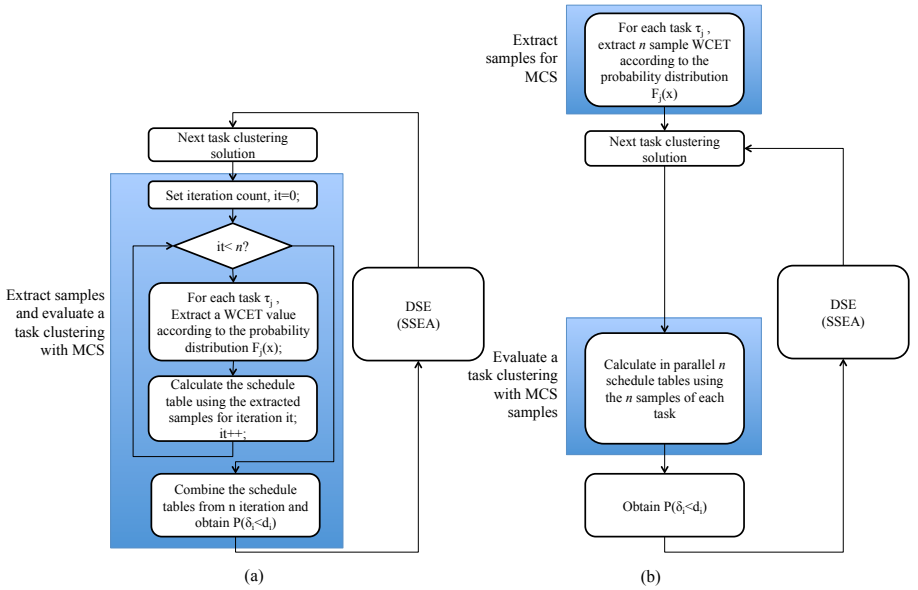


Figure 3.4: Steps for performing design space exploration with MCS (a) Reordered steps for performing design space exploration with MCS (b)

assigned to different hardware resources (PEs and CEs). When tasks and messages are assigned to the same hardware resource and there is a contention of the same layer, their execution order is set according to their priority.

According to MCS, for each task, we extract n random samples according to the normal distribution modeling the WCET of the task. For the messages we have arrays that contain n equal values, i.e. the transmission time associated to the message (C_{m_g}). In Figure 3.5, there is an example. Please note that even if we use the same naming convention for the stochastic variable C_j and for the array of n-elements that are modeling the probability distribution of the variable, we use the boldface formatting to indicate the n-element array (the same convention applies to the WCET of a message and its corresponding n-element array).

To speed up the *MCS*, we move the random generation of *n* WCET values outside of the DSE. Additionally, instead of iterating the calculation of the schedule table n times, we execute the schedulability analysis a single time on arrays of n samples containing the extracted WCET so that the layer subdivision is calculated only once. Figure 3.4b depicts the steps of the MCS for speeding up its execution.

During the schedulability analysis, we combine these arrays of n-elements using **+** (sum) and **max** (maximum element selection) operations. Each operation is performed

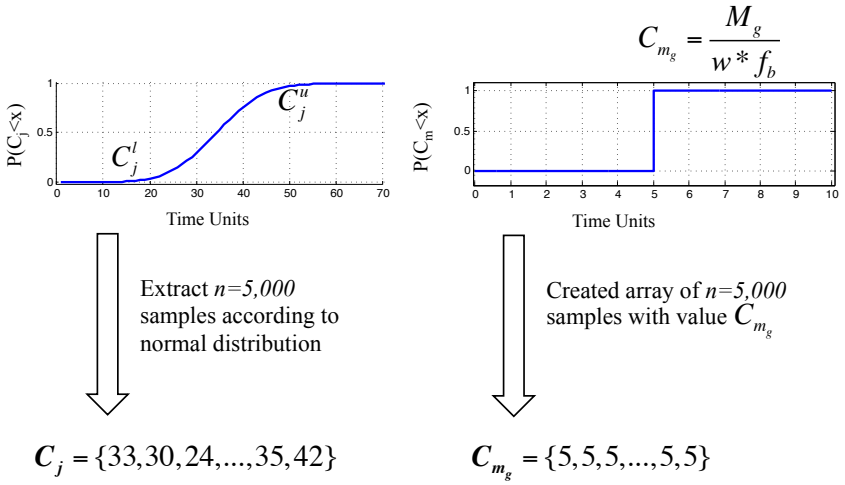


Figure 3.5: Extraction of n samples to build the arrays $\boldsymbol{C_j}$ and $\boldsymbol{C_{m_g}}$

element by element on the arrays. Using these arrays and the layer subdivision (indicated by the dotted lines in Figure 3.6c), we calculate the starting time $\boldsymbol{t_j^s}$ ($\boldsymbol{t_{m_g}^s}$) and finishing time $\boldsymbol{t_j^f}$ ($\boldsymbol{t_{m_g}^f}$) of each task τ_j (and message m_g). They are obtained combining the $\boldsymbol{C_j}$ of the tasks and $\boldsymbol{C_{m_g}}$ of the messages. Therefore, each starting and finishing time is also a stochastic variable which probability distribution is modeled by an n-element array. The starting and finishing time of each task and message are calculated for each layer, starting from the first layer down to the last one.

The starting time $\boldsymbol{t_j^s}$ of a task τ_j is given by the maximum of the finishing times of all the tasks which τ_j depends on (**max** operation). If τ_j has no data dependencies or hardware resource contention, its starting time is zero. The finishing time $\boldsymbol{t_j^f}$ of τ_j is given by the sum (**+** operation) of the starting time and of the WCET of the task, i.e., $\boldsymbol{t_j^s + C_j}$. Once we have the starting and finishing time of each task, we consider all the sink tasks, i.e. tasks without successors. The array with the maximum finishing time among the sink tasks in an application A_i is used to extract the CDF modeling the performance of the entire application and corresponds to δ_{A_i}: given the final array of n elements, we build its histogram, we normalize it (obtaining the PDF) and we integrate it to obtain the CDF δ_{A_i}. Finally δ_{A_i} is evaluated according to the application deadline d_i and returns the probability p_i of having a schedulable solution. Figure 3.6c depicts the computations performed at each layer.

We assume that the WCET CDFs of the tasks are independent. Therefore, clustering tasks together has no effect on their CDFs. We are using this approach in the very early phases of the design, when the ASIPs have not been synthesized. As a consequence,

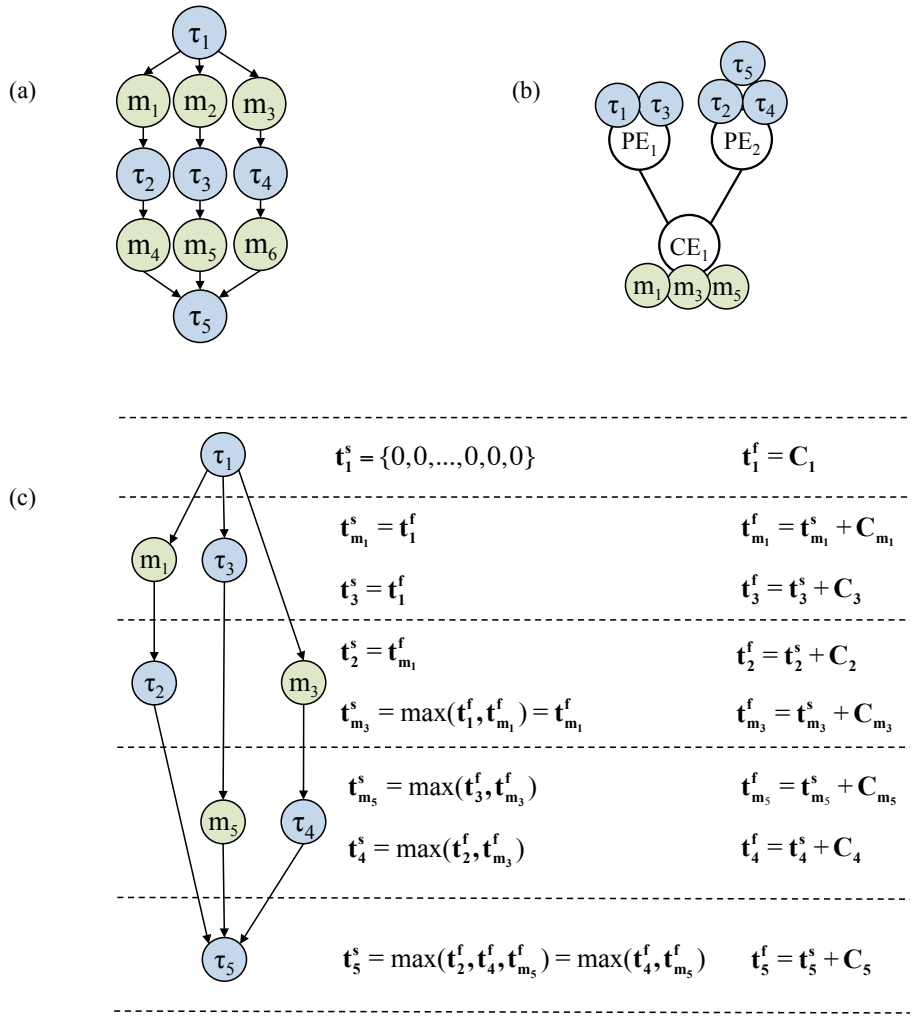


Figure 3.6: Example of application A_i (a), of task clustering solution for A_i (b) and of δ_{A_i} calculation (c) [69]

we can assume that each ASIP micro-architecture will contain the necessary hardware components to provide the expected WCET for each task. In Chapter 8, we propose an extension of the DSE that also considers the area of each ASIP and in that case we also analyze the impact of the task clustering on the WCET of the tasks.

3.3.2 Comparison of task clustering solutions

Given the δ_{A_i} of multiple task clustering solutions, we can compare them calculating their probability of meeting the deadline d_i. Let us consider the δ_{A_i} of two task clustering solutions: Sol_1 and Sol_2 (Figure 3.7a). Sol_1 has a probability p_i of 0.82 while Sol_2 of 0.03. This indicates that the first task clustering solution is much more likely to meet the application deadline $d_i = 65$ when the platform is designed. Additionally, during DSE, we may need to compare clustering solutions with the same probability (e.g. Figure 3.7b where both solutions have $p_i \sim 1$ for a deadline $d_i = 80$). To discriminate between these solutions we consider the number of PEs used, and select the solution with the smaller number of processors. If also this value is the same, then we use the inverse of the CDF, the quantile function $\delta_i^{0.5} = P^{-1}(p_{0.5})$ (defined in Equation 2.4): we select the solution Sol_{id} that has the *smallest* WCET $\delta_i^{0.5}$ at a probability $p_{0.5} = 0.5$. In the example in Figure 3.7b, the value of quantile function at $p_{0.5}$ is 62 time units for Sol_1 and 68 time units for Sol_2. Therefore during our exploration we will select Sol_1.

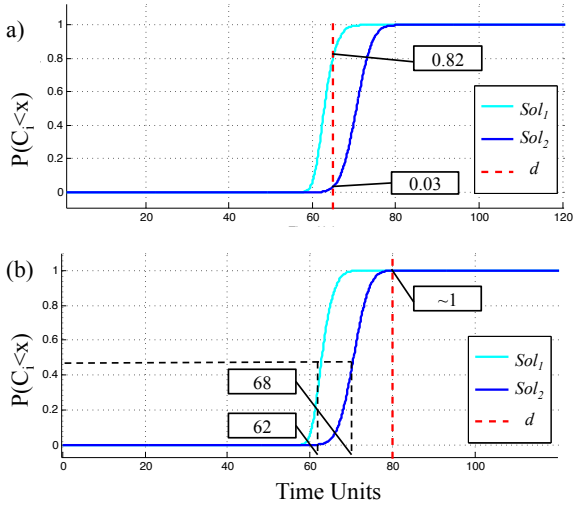


Figure 3.7: Comparison between task clustering solutions

3.3.3 Evolutionary Algorithm

We use a Steady State Evolutionary Algorithm (SSEA) [15] to decide the clustering of tasks. SSEA takes as input the applications (including the uncertainty model) and the CP model of the initial platform (with the maximum number of ASIPs, PC_{max}).

The algorithm returns a task clustering solution, which maximizes the schedulability probability p for all applications, i.e., $p = (\sum p_i)/N_A$ where N_A is the number of applications, under the given cost constraint PC_{max}. When multiple solutions have the same probability we select the one with the smallest quantile function $P^{-1}(p_{0.5})$ (we use the quantile function of the CDF modeling the total performance of all applications) as presented in 3.3.2.

SSEA is inspired from the process of natural evolution, where a set of solutions is called a *population* and each solution is encoded using a string called a *chromosome*. The population is evolved by performing recombination and mutation, and part of the population is replaced by the *offspring* population, which has better *fitness* according to the cost function. SSEA has been chosen as it is suitable for optimization problems in which the computation of the cost function is time-consuming (a small portion of the population is replaced at each new generation). The algorithm works by adding the offspring of the individuals selected from each generation to the pre-existing one, so individuals are retained between generations. The implemented SSEA is described in Algorithm 1.

A chromosome represents a task clustering solution. It is defined as an array of tasks and messages; the value of each element (*gene*) represents the identifier of the PE or CE to which the tasks and messages are respectively assigned. We used a two-point crossover operator [15]. The mutation operator is used to randomly vary the PEs and CEs to which the tasks and messages are assigned. The parameters used for the execution of the SSEA are: crossover probability P_c, mutation probability P_m and the population size Pop. SSEA finishes when a given time-limit has been reached. The tuning of these parameters has been done running multiple executions of the algorithm with different synthetic applications.

3.3.4 Example of schedulability analysis with UM

In this section, we use a small real case study (Figure 3.8a) to demonstrate the effectiveness of our schedulability analysis and DSE. Without the use of our UM, a designer willing to identify the proper task clustering for its input applications, can use a template ASIP micro-architecture. The designer can characterize the WCET of each task τ_j executing the task on the template processor. We denote this WCET as reference WCET, C_j^{ref} and this design approach as straightforward method. We use the SFM as a reference to compare the results obtained with our UM approach.

The task graph in Figure 3.8a has a source task τ_{so}, which is a dummy task with WCET equal to zero and which is used only for starting the remaining tasks. Also the output messages m_{so} of τ_{so} are dummy messages with zero bit of data associated. Table 3.1 contains the upper and lower bounds for the UM and C_j^{ref} for the SFM for the re-

Algorithm 1 - Steady State EA

1: $P :=$ Generate Initial Population
2: **repeat**
3: $Sol_1, Sol_2, Sol_3, Sol_4 :=$ Random Select 4 different clustering solution from P
4: $p :=$ Generate Random Probability
5: $best_1 :=$ Select best clustering solution between Sol_1 and Sol_2
6: $best_2 :=$ Select best clustering solution between Sol_3 and Sol_4
7: **if** $p > P_c$ **then**
8: $o_1, o_2 :=$ Apply two-point crossover to $best_1$ and $best_2$
9: **else**
10: $o_1 := best_1$
11: $o_2 := best_2$
12: **end if**
13: $o_1' :=$ Apply mutation to o_1 with probability P_m
14: $o_2' :=$ Apply mutation to o_2 with probability P_m
15: $r_1, r_2, r_3, r_4 :=$ Random Select 4 different clustering solution from P
16: $worst_1 :=$ Select worst clustering solution between r_1 and r_2
17: $worst_2 :=$ Select worst clustering solution between r_3 and r_4
18: Replace $worst_1$ with o_1' in P
19: Replace $worst_2$ with o_2' in P
20: **until** $finished()$

maining tasks in Figure 3.8a. These values are derived as follows. For each task, we considered a simple functionality, consisting of a loop and operations such as multiplication and addition. We used the SH ASIP toolchain [55] to design the ASIPs. We implemented a three-issue slot VLIW ASIP, and we ran the tasks to obtain their WCETs. These WCET[2] values were used as the reference C_j^{ref}. Furthermore, we varied the micro-architecture of this ASIP to obtain two extreme ASIP configurations. The WCETs on the slowest ASIP thus obtained were considered the upper bound C_j^u, whereas the WCETs on the fastest ASIP designed were considered the lower bound C_j^l. The obtained values are presented in Table 3.1. We considered that the platform had at most two ASIPs ($PC_{max} = 2$).

Each platform solution (derived from a certain task clustering) was evaluated using the schedulability analysis from Section 3.3, which gave the probability p of a solution to be schedulable after implementation. We performed an exhaustive DSE as the design space for this example is limited and we want to consider all possible task clustering solutions. The best clustering solution obtained with UM is shown in Figure 3.8b,

[2]We are aware that the value obtained from the execution of the tasks on the ASIP is not a real WCET, which is a theoretical upper bound determined through analysis, but we believe this value is a good approximation for our experiments: the loop bounds are known at compile time and the input data are hard-coded in the tasks' code.

Table 3.1: C values for the motivation example (in μs)

C	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7
C_j^u	702	3437	8801	702	702	702	3437
C_j^l	450	180	300	450	450	450	180
C_j^{ref}	602	3237	400	602	602	602	3237

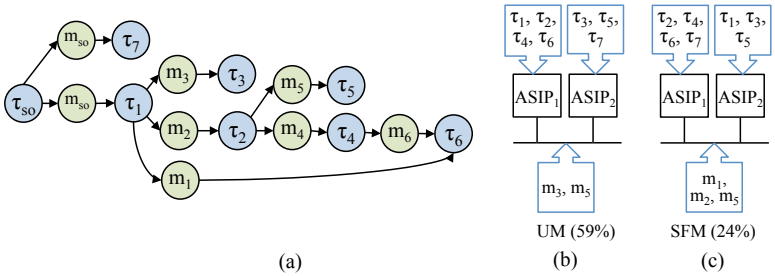


Figure 3.8: Comparison between UM and SFM approaches

having a $p = 59\%$. Then, we performed an exhaustive DSE of all possible clustering solutions with SFM, aiming at minimizing the schedule length, considering the given C_j^{ref}. The task clustering obtained with SFM is shown in Figure 3.8c. In order to compare the solutions obtained with the UM and SFM, we calculated the probability p of the task clustering solution, found by the SFM, when evaluated using the UM approach. Thus, p for the solution with SFM is 24%.

To validate the conclusions of the comparison between UM and SFM, we implemented the platform solutions in Figure 3.8b and 3.8c, produced by UM and SFM, respectively. Next, we determined the WCETs C_j of each task τ_j on their respective ASIPs using SH cycle-accurate simulator. Then, we obtained the optimal schedule lengths for the two cases. The schedule length in the case of the UM platform solution is $1,955\ \mu s$, whereas for SFM is $2,275\ \mu s$. For a deadline of $2,000\ \mu s$, UM solution is schedulable, while SFM solution is not.

This example suggests that if a UM solution has higher chances to be schedulable compared to a SFM solution according to our evaluation (Section 3.3.1), this is also true in the final implementation using SH tools. The comparison of the two approaches shows that with our UM approach, we are able to identify a solution that has a higher probability to be schedulable, once it is implemented.

3.4 Summary

In this chapter we presented the problem formulation, i.e., the definition of a multi-ASIP platform through the schedulability analysis of different task clustering solutions. Our schedulability analysis is based on the UM. The probability distribution of the tasks and the WCET of the messages are combined to calculate the probability of a task clustering solution to meet the application deadline. Moreover, we explained how our schedulability analysis could be used to compare different task clustering solutions and we described the DSE algorithm used to explore the design space of the solutions. Furthermore, we presented an example that shows the application of our DSE with UM; the result obtained is verified implementing the multi-ASIP platforms, found by our DSE, using the SH ASIPs.

CHAPTER 4

Experimental evaluation with Task Graph model

This chapter is dedicated to the evaluation of our DSE with UM (part of the material described in this chapter has been published in [69]). It is organized as follows. In Section 4.1 we demonstrate the effectiveness of our UM applying it to synthetic and real-life benchmarks and comparing it with the SFM (described in Section 3.3.4). Additionally, in Section 4.2, we consider a Motion JPEG (MJPEG) encoder and after running our evaluation, we design the multi-ASIP platform implementing the task clustering suggested by our DSE (using SH technology). For comparison, we also implement different task clustering solutions. Moreover, we perform a sensitivity analysis on the upper and lower bounds used to build the UM of each task.

The tools for the experimental evaluation have been developed in MATLAB for prototyping and then in Java for the final implementation. We run the DSE tools on a machine with Intel Core i7 CPU (2 GHz) and 8GB of RAM.

4.1 Comparison of DSE with UM and SFM

For the experimental evaluation of our DSE with UM, we consider both real-life and synthetic benchmarks. For the real-life benchmarks, we use two subsets of applications

from the Embedded System Synthesis Benchmark Suite (E3S), version 0.9 [2], taken from the telecommunication (*telecom-cords*, *TLC*) and automotive/industrial (*auto-indust-cords*, *IND*) domain and two synthetic case studies (*Synth*), which represent a smaller and a wider example. The case study *Synth 1* is the same presented in 3.3.4. The details of the case studies, in terms of number of applications and tasks are presented in Table 4.1. For each case study, we have a deadline d (column 4) and a platform constraint, PC_{max} (column 5). We consider that all applications in a case study, have the same deadline.

Table 4.1: Comparison of UM and SFM [69]

Case Study	No. of Apps.	No. of Tasks	d (ms)	PC_{max}	*UM*		*SFM*	
					p	ASIPs	p	ASIPs
TLC	4	10	5.35	3	71%	3	55%	3
IND	4	13	5.25	4	70%	4	59%	4
Synth 1	2	7	5	2	59%	2	24%	2
Synth 2	10	22	5.6	5	59%	5	49%	5

For each real-life benchmark, we consider the WCET value in the E3S benchmark as the reference WCET C_j^{ref}, and we scale this value to obtain the lower (C_j^l) and upper (C_j^u) bounds. For *Synth 1* we used the values in Table 3.1 and arbitrary values for *Synth 2*.

We ran the Steady-State Evolutionary Algorithm (SSEA) with the UM from Section 2.3 on each of the four benchmarks and we obtained a task clustering solution (we ran an exhaustive DSE for *Synth 1*). The probability p of each benchmark to be schedulable with the obtained task clustering solution is presented in Table 4.1, column 6, which also presents the number of ASIPs used (column 7). The overall schedulability probability p is calculated as an average of the probability p_i of each application A_i in the benchmark.

Together with the UM results, Table 4.1 also presents the SFM (see Section 3.1), which uses a reference value for the WCET C_j^{ref} of each task τ_j. This value is used inside the SSEA optimization instead of the stochastic WCETs. We use this value for the calculation of the end-to-end response time (δ_{A_i}). This is what a designer would do following a more traditional design approach and if a WCET uncertainty model would not be available.

The parameters used for the execution of the SSEA are: $P_c = 30\%$, $P_m = 10\%$ and $Pop = 100$. The execution time has been set to 1 hour.

The results in Table 4.1 show that the UM is able to obtain better results, i.e. a higher the probability of finding schedulable implementations, compared to SFM. In fact, using our proposed UM, we can take into account a bigger number of ASIP micro-architecture configurations during the system-level DSE for multi-ASIP platforms. In Table 4.2 there are the task clustering solutions found by the DSE with the UM and with the SFM for each case study (in the table, a task is indicated as $\tau_{i,j}$, where i indicates the application A_i and j is the task id). We recall that for the case study *Synth 1*, it was possible to implement the ASIPs and verify that the difference in the p returned by UM and SFM is reflected in the final schedule table after the platform implementation. This shows that our DSE with UM is able to lead to good final implementations: a scheduling length of 1.955 ms for the clustering solution found with the UM versus the 2.275 ms of the one found with the SFM.

Table 4.2: Clustering results for real-life and synthetic case studies

ID	Type	PE_1	PE_2	PE_3	PE_4	PE_5	B	PE nr.	p
TLC	*SFM*	$\tau_{2,3}, \tau_{3,1}, \tau_{4,1}$	$\tau_{1,1}, \tau_{1,4}, \tau_{2,2}, \tau_{2,4}$	$\tau_{1,2}, \tau_{1,3}, \tau_{2,1}$	-	-	b_{32}^{10}	3	0.55
	UM	$\tau_{1,1}, \tau_{2,2}, \tau_{2,4}, \tau_{3,1}$	$\tau_{1,2}, \tau_{2,1}$	$\tau_{1,3}, \tau_{1,4}, \tau_{2,3}, \tau_{4,1}$	-	-	b_{32}^{10}	3	0.71
IND	*SFM*	$\tau_{1,1}, \tau_{1,3}, \tau_{1,4}, \tau_{1,5}, \tau_{1,6}, \tau_{1,7}$	$\tau_{2,2}$ $\tau_{2,3}, \tau_{2,4}, \tau_{4,1}$	$\tau_{1,2}, \tau_{5,1}$	$\tau_{2,1}, \tau_{3,1}$	-	b_{16}^{10}	4	0.59
	UM	$\tau_{1,1}, \tau_{1,3}, \tau_{2,3}, \tau_{1,5}, \tau_{1,7}$	$\tau_{1,6}, \tau_{2,1}$ $\tau_{1,4}, \tau_{2,4}, \tau_{1,2}$	$\tau_{4,1}, \tau_{5,1},$	$\tau_{2,2}, \tau_{3,1}$	-	b_{16}^{10}	4	0.70
Synth 1	*SFM*	$\tau_1, \tau_2, \tau_4, \tau_6$	τ_3, τ_5, τ_7	-	-	-	b_{16}^{10}	2	0.24
	UM	$\tau_2, \tau_4, \tau_6, \tau_7$	τ_1, τ_3, τ_5	-	-	-	b_{16}^{10}	5	0.59
Synth 2	*SFM*	$\tau_{1,1}, \tau_{1,4}, \tau_{5,1}, \tau_{5,2}, \tau_{5,3}$	$\tau_{2,1}, \tau_{3,1}, \tau_{5,4}, \tau_{8,1}$	$\tau_{1,3}, \tau_{4,1}, \tau_{6,2}, \tau_{6,3}, \tau_{7,1}$	$\tau_{2,3}, \tau_{6,2}, \tau_{9,1}, \tau_{10,1}$	$\tau_{1,2}, \tau_{2,2}, \tau_{2,4}, \tau_{6,1}$	b_{16}^{10}	5	0.49
	UM	$\tau_{1,1}, \tau_{2,2}, \tau_{4,1}, \tau_{2,4}$	$\tau_{6,1}, \tau_{6,2}$	$\tau_{1,2}, \tau_{3,1}, \tau_{6,3}, \tau_{6,4}, \tau_{7,1}, \tau_{9,1}$	$\tau_{1,4}, \tau_{5,1}, \tau_{5,2}, \tau_{5,3}, \tau_{5,4}, \tau_{10,1}$	$\tau_{1,3}, \tau_{2,1}, \tau_{2,3}, \tau_{8,1}$	b_{16}^{10}	5	0.59

4.2 Experimental evaluation of DSE with UM using SH tools

In this section, we use a real case study from the multimedia domain to demonstrate the correctness of the UM and of our macro-architecture DSE: a motion JPEG (MJPEG) encoder [60]. We implemented the task clustering solution suggested by our exploration using SH technology to validate our results. We applied the design flow shown in Figure 4.1 that is a subset of ASAM design flow [7]. ASAM is an European research project part of ARTEMIS Joint Undertaking. Its purpose is to implement a semi-automatic design flow and a toolchain for the design of multi-ASIP platforms, providing tools for macro-architecture, communication architecture and micro-architecture DSEs. More details about ASAM design flow and the integration of our macro-architecture DSE to the flow are available in Chapter 7.

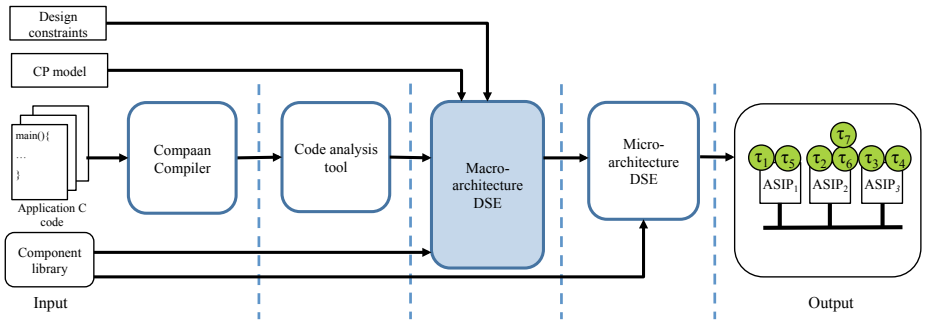


Figure 4.1: Semi-automatic design flow for multi-ASIP design using the UM

Starting from the sequential C code of the MJPEG encoder, we implemented a multi-ASIP system using our DSE with UM to determine the best task clustering solution. As support to our experiment, we used external tools for the application partitioning (Compaan compiler [1]) and for the code analysis of the tasks (Phase 1 of ASAM micro-architecture DSE[1] [43]). For the definition of the micro-architecture of a single ASIP we used Phase 2 of ASAM micro-architecture DSE[1] [43], and for the implementation of the entire platform, we used SH tools for ASIP design. SH provides proprietary languages for the hardware description of the platform together with a cycle-accurate simulator that returns the performance of the application running on the multi-ASIP platform. Additional details about the external tools are provided in Chapter 7. The design flow in Figure 4.1 requires as inputs:

- the sequential C code of the application A_{MJPEG}

[1]Tool developed by TU/e - Technical University of Eindhoven

- the deadline d_{MJPEG} of the application
- the desired working frequency f for the ASIPs
- an initial platform description (CP model) with the corresponding platform cost (PC_{max}) and bus types that we want to explore (Figure 4.5a)

For this case study, we limited our communication architecture exploration to a single bus b_{32}^{f} (i.e. a 32 bit width bus with the same frequency f of the multi-ASIP system). In this way, we could validate our solution against the actual implementations of the ASIPs and platform that we had available (this was not a limitation of SH but of the micro-architecture DSE tools that we used[1]). We considered the elaboration of 15 frames and a desired throughput of 25 frames per second (fps) that gave a deadline $d_{MJPEG} = 0.6s$ for the elaboration of the 15 frames. We set $PC_{max} = 3$.

We used Compaan tool [1] for the partitioning of the application into tasks. Compaan is a commercial tool that elaborates sequential C code (opportunely modified for the tool requirements) and builds the corresponding Kahn Process Network (KPN) [50]. The output of Compaan compiler for the MJPEG encoder is depicted in Figure 4.2a. Each actor in the KPN model has a function associated (e.g., *ND_3* executes the kernel function *mainDCT*).

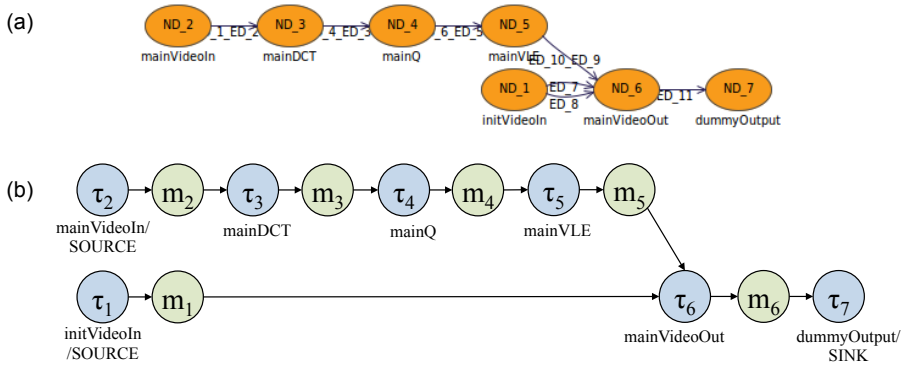


Figure 4.2: (a) KPN and (b) TG models for MJPEG encoder

We extracted the task graph of the MJPEG encoder (Figure 4.2b) from Compaan KPN model. We transformed each KPN actor into a task and each edge into a message; each message has associated the number of bits of data to transfer.

When there are multiple edges between the same couple of actors (e.g., in Figure 6.2a there are two edges between actors *ND_1* and *ND_6*), they are substituted by a single message. The presence of multiple edges can fall under one of the following cases:

either these edges represent mutually exclusive data transfers, i.e. at each completion of the actor producer, only one of these edges is executed (*case A*), or all edges are executed at the completion of the actor producer (*case B*). In *case A*, we set the size of the message to the size of the edge with the biggest amount of data to transfer, so as to consider the worst case scenario. In *case B*, we set the size of the message to the sum of the data to transfer on all edges. The two edges between source actor *ND_1* and target actor *ND_6* fall under *case A*, so we selected the biggest one.

The application model obtained from the KPN of Compaan corresponds to the elaboration of one frame of data. In our schedulability analysis, for the elaboration of 15 frames, we considered 15 repetitions of the task graph in Figure 4.2.

Then we used the code analysis tool described in the ASIP DSE (Phase 1) of [43] to determine the upper and lower bounds (C^l and C^u) for each task in the application. The code analysis tool profiles the application code (using LLVM compiler [64]) and, for each task, it estimates the number of cycles required by a sequential execution (C^u) and by a parallelized execution (C^l) of the code. The code analysis tool bases its estimation on the intermediate representation generated by LLVM, however the number of cycles of each instruction are the same cycles used by SH cycle-accurate simulator. The tool returns the number of cycles of each task for the entire execution of the application.

For the case study, we divided the values returned by the code analysis tool by 15, as, for building the schedule, we needed the number of cycles of a single execution of the tasks. Therefore, we obtained the average numbers of cycles for each task firing that are summarized in Table 4.3. The source and sink tasks of the MJPEG encoder (i.e. tasks that have no input edges and tasks without output edges, respectively) are used for data initialization (i.e. for writing the input data into a local or external memory of the multi-ASIP platform that we want to design), and for providing feedback to the user about the completion and exit status of the application. For this reason, we considered their execution time equal to zero and did not assign them to any ASIPs.

Then we built the CDF of each task using the estimated C^l and C^u and the input frequency $f = 166MHz$ (Figure 4.3). The CDF is obtained using Equations 2.3, 2.5 and 2.2 that allows calculating the mean, variance and finally the CDF. The amount of data expressed in bits of each message is shown in Table 4.4. We calculated the transmission time of the messages on the bus considering the bus b_{32}^f.

Then we can executed our macro-architecture DSE that performs the schedulability analysis of different task clustering solutions. We ran the SSEA for $200\ s$. The output of our DSE is shown in Figure 4.4: we found a solution that has $p_{MJPEG} \sim 1$ to meet the deadline and that uses two ASIPs. The first row of Table 4.5 (columns 2-4) summarizes the outcome of our exploration: the task clustering solution, the probability of the application to meet the deadline and the quantile function value at a probability of 0.5. To verify our result, we used the Phase 2 of the micro-architecture design tool [43]

Table 4.3: C values for MJPEG encoder (average number of cycles for a single iteration of the task)

C	$mainDCT$	$mainQ$	$mainVLE$	$mainVideoOut$
C_j^u	4774994	591698	1719197	130584
C_j^l	1713106	426834	1089013	105205

Table 4.4: Message sizes (in bits) for MJPEG encoder

m_1	m_2	m_3	m_4	m_5	m_6
128	1048576	1048576	1048576	524288	32

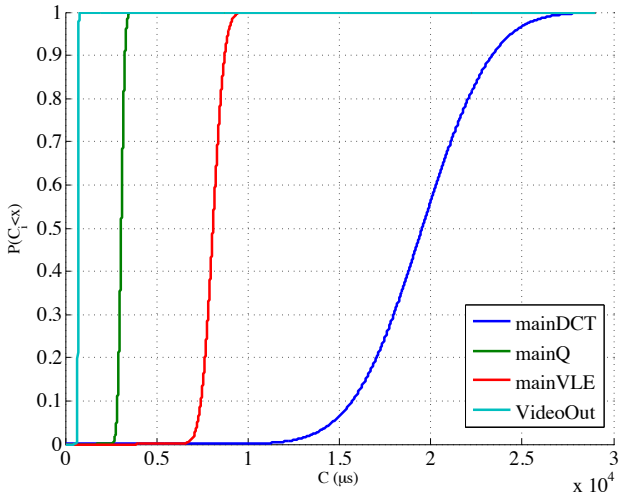


Figure 4.3: Cumulative distribution functions for the tasks of the MJPEG encoder application (with $f = 166MHz$)

and we obtained a description of the micro-architecture of the two-ASIPs.

The micro-architecture DSE defines a single ASIP given a task cluster and a library of predefined issue slots, ISs, built using SH tools. The ISs available in the library represent slices of an ASIP: they contain a RF, multiple functional units and an optional data memory. Moreover, there is a default IS that is always included and contains the program counter, the instruction memory, a default data memory and a fixed number of FIFO ports. The micro-architecture DSE requires the polyhedral model [10] of each cluster of tasks: the code inside a task needs to be specified as affine nested loops [20].

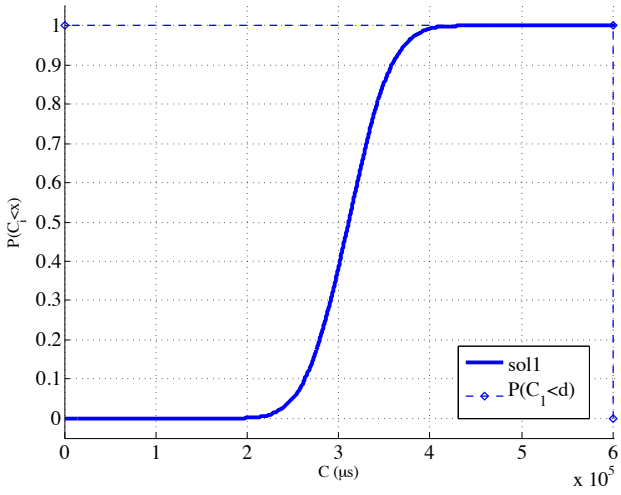


Figure 4.4: Results from the macro-architecture DSE for MJPEG encoder

Then the micro-architecture DSE optimizes this model using loop-based transformations and it obtains the instruction level parallelism inside the loop body of each task (the loop body contains the kernel function). The instruction level parallelism is used to determine the number and type of ISs in the ASIP micro-architecture. Moreover, the micro-architecture DSE uses the input and output data that are consumed and produced by each task, respectively, to size the data memories of the ISs.

For the two task clusters found by our macro-architecture DSE, the micro-architecture DSE defined two ASIPs, each of them with 3 ISs (including the default one). After obtaining the micro-architecture description of the ASIPs, using SH tools, we implemented the two-ASIP platform and we mapped the application code to the ASIPs as suggested by our DSE. The sink and source tasks are mapped to a *host processor*. A SH platform, for simulation purposes, requires the insertion of a host processor that is used for uploading the C code of the tasks onto the ASIPs, for initializing the memories and for starting the code execution (in Section 7.2.4 we provide additional details). The resulting platform has two ASIPs, each of them with 3 ISs; they are connected through a system bus for the exchange of data. Additionally, we added two FIFOs between the two ASIPs that need to exchange data. The FIFOs are used only for synchronization purposes. For the synchronization of the ASIPs with the host processor, we needed to provide the host with the access to the FIFO ports of the ASIPs. The host does not have any FIFO ports and the access to the FIFO ports of the ASIPs is guarantee through a hardware block called FIFO adapter. We insert a FIFO adapter for each each ASIP (additional details are available in Section 7.2.4). Figure 4.5b contains a schematic of the designed platform.

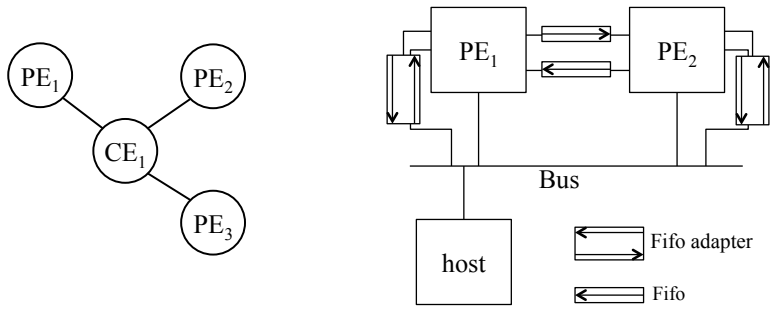


Figure 4.5: (a) CP model of the initial platform and (b) designed platform for MJPEG encoder case study

For consistency with our schedulability analysis, we also tuned the application code so that the processors were communicating through a message-passing paradigm. For example, in our case study, task *mainDCT* is assigned to PE_1 and is producing data for task *mainQ* that is assigned to PE_2: *mainDCT* reads the input data from the local memory of PE_1 and, after elaboration, it writes them to the local memory of processor PE_2 where *mainQ* can read them. Moreover, in our schedulability analysis, we added some cycles of offset for modeling the time required for starting the execution of the tasks on the ASIPs, for modeling the synchronization time (access to the FIFOs), and also for considering additional bus parameters as the hand-shake time to gain access to the bus and the setup time for transfer the data.

In columns 5-6 of Table 4.5 there are the number of execution cycles obtained with SH simulator (*sim*) and the corresponding time in μs (at a frequency $f = 166MHz$). After the design of the multi-ASIP platform, we could also verify that our implementation was schedulable (column 7).

It is important to note that our DSE works by comparison: we can evaluate different task clustering solutions and determine which one has highest chances to produce a schedulable implementation once the final platform is available, but we are not guaranteeing the schedulability of the application. We use our approach in the very early phases of the design when there is no implementation available for the platform and it can help the designer in determining the platform composition and the partitioning of the application. For this reason, we also considered other clustering solutions: we evaluated them with our DSE and we implemented and simulated them with SH tools. The results that we obtained from our schedulability analysis are shown in Figure 4.6 and in Table 4.5.

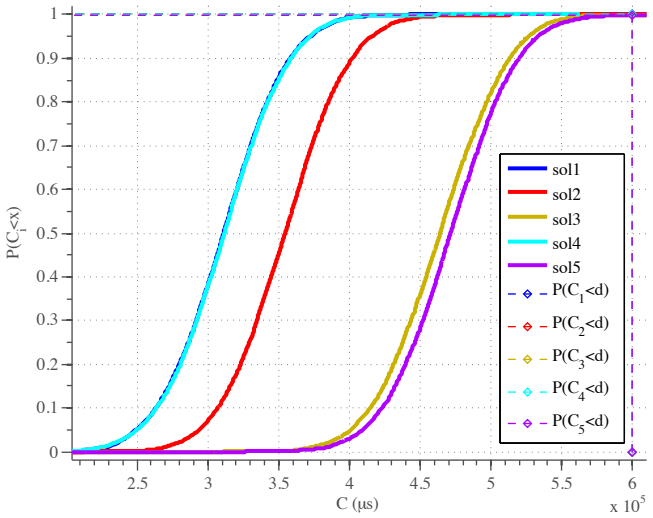


Figure 4.6: Comparison of the CDF of different clustering solutions for MJPEG encoder

We tested those task clustering solutions with two ASIPs (Sol_2 and Sol_3) that are a fair alternative to the clustering solution found by our DSE. Additionally, we verified the performance of a solution with a single processor (Sol_5) and the performance that can be achieved using three ASIPs. We selected these task clustering solutions considering that $mainDCT$ is the task with the highest number of cycles, and, therefore, there should be a dedicated ASIP for its execution. Additionally, it is better to cluster successive tasks on the same ASIP: for example, clustering $mainDCT$ and $mainVLE$ together (and $mainQ$ and $mainVideoOut$ together) is inconvenient as it forces multiple exchanges of data between the two ASIPs (this implies also that the processors will stall waiting for the data). Moreover, we verified that using an additional processor (Sol_4), there are no improvements in the performance. We wanted to verify if having a dedicated processor for $mainDCT$ and splitting $mainVLE$, $mainQ$ and $mainVideoOut$ into two processors we could speed up the execution. We verified that $mainDCT$ is still the task defining the speed of the entire system and we obtained a higher number of cycles due to the additional communication and synchronization time due to the extra processor. This result are confirmed by our schedulability analysis and the simulation with SH tools.

The clustering solutions Sol_2, Sol_3 and Sol_5 are worst than Sol_1 that is the one produced by our exploration. Sol_4 has a CDF very similar to the one of Sol_1 (in Figure 4.6 they are overlapping); however, Sol_4 uses three processors, so Sol_1 is preferred by our DSE; it also has a higher quantile value. This is reflected in the simulation result af-

ter implementation that, for Sol_4, shows a higher number of cycles than Sol_1. Also for Sol_2, Sol_3 and Sol_5 there is a qualitative match between the results obtained with our estimation with UM and the results obtained after implementation with the cycle accurate simulator from SH. Most of these solutions have similar probability (~ 1). Therefore, the quantile value (column 4 of Table 4.5) is evaluated. The clustering solutions that are better with our estimation, are also better after implementation. We can sort the solutions in Table 4.5 based on their likelihood to be schedulable (according to our DSE) and we can find that the ordering is maintained after implementing and simulating them using SH tools.

4.2.1 Additional considerations

The code analysis tool [43] used for the estimation of the upper and lower bounds does not return a theoretical estimation of the WCET. It returns an estimated number of cycles of a profiled execution of the MJPEG encoder. Therefore, we ran the tool with multiple input data (frames with same size, but different content) and we took the ones producing the highest estimated execution time. We used this input to set the upper and lower bounds.

We consider that these values are good enough to verify our UM as the variability in the estimated execution time given by the different input data is 2%. Additionally, we used the two-ASIP platform associated to the best task clustering (Sol_1) to execute the MJPEG encoder with the same set of input data. The cycle-accurate simulator of SH is returning very similar results with the different input stimuli. The execution cycles vary from 90,514,748 to 90,915,481 cycles, i.e. less than 1%. In general the performance of a VLIW architecture can be predicted more easily than a superscalar processor [100]. In a VLIW architecture, the compiler is in charge of scheduling the instruction statically and there is no run time optimization. Additionally, the ASIPs that we are considering in our case study have no cache memory and this also makes the performance more predictable. Taking into account these elements, we consider the execution time and the estimation of the upper and lower bounds quite reliable and sufficient to demonstrate our approach even if they not correspond to a theoretical WCET.

4.3 Accuracy of C_j^l and C_j^u

In this section we analyze the impact of the upper and lower bounds on the results of the DSE. Our UM assumes that the designer or a code profiling tool provide the upper and lower bounds of the WCET of each task; therefore, we verify how much this estimation

Table 4.5: Comparison of clustering solutions for MJPEG encoder

Sol_{ID}	Clusters			$P(\delta_{A_{MJPEG}} < d)$	$\delta^{0.5}_{A_{MJPEG}} = P^{-1}(p_{0.5})\ (\mu s)$	*sim* (cycles)	*sim* (μs)	sched
	PE_1	PE_2	PE_3					
1	mainDCT	mainQ, mainVLE, mainVideoOut	-	~ 1	311200	90915481	547683.62	yes
2	mainDCT, mainQ	mainVLE, mainVideoOut	-	~ 1	354700	103584916	624005.51	no
3	mainDCT, mainQ, mainVLE	mainVideoOut	-	0.99	464200	142745646	859913.53	no
4	mainDCT	mainQ, mainVLE	mainVideoOut	~ 1	311800	91146918	549077.82	yes
5	mainDCT, mainQ, mainVLE, mainVideoOut	-	-	0.99	471500	143166046	862446.06	no

should be accurate. We performed a sensitivity analysis on the C_j^l and C_j^u values of the tasks.

First, we used the case study *Synth 1*, which task graph is described in Figure 3.8. For each task, we considered variations from $\pm 1\%$ to $\pm 10\%$ of C_j^l and C_j^u values. In particular, we considered a total of 60 cases in which all tasks or a subset of them are suffering variations. These changes in the bounds are reflected in slightly different input CDFs for our DSE. We ran our DSE for each of these cases, and we obtained the same clustering solution, which suggests that our DSE is not sensitive to small variations in the WCET estimation provided by the designer or profiling tool.

Second, we performed this check also on the MJPEG encoder case study (Section 4.2). We verified how accurate the C_j^l and C_j^u found by the code analysis tool [43] are. We compared the upper and lower bound values estimated by the code analysis tool (C_j^l and C_j^u) with the number of cycles obtained for the execution of the entire application on a single ASIP using SH simulator. We used an oversized ASIP with a large number of ISs to theoretically exploit all instruction level parallelism of the application (the parallelism during execution can be lower depending on the compiler optimization). In particular, we compared the estimated (C_j^l and C_j^u) and simulated (*sim*) values of *all iterations* of the tasks.

The values obtained for MJPEG encoder are summarized in columns 2 to 4 of Table 4.6. The C_j^l, C_j^u and *sim* for each task τ_j are quite different and the *sim* values are not included in the range $[C_j^l, C_j^u]$ as expected (when we compare the C_j^l with the results obtained with simulation, we have relative errors up to 67%). These differences can be justified considering some inaccuracy in the estimation done by the code analysis tool. The analysis of the accuracy of the code analysis tool is provided in [42]. For the case studies described in [42], there is less than 10% underestimation for the evaluated number of cycles compared to the simulated one. In our case, the bigger differences between estimation and simulation results derive from the more complex application code to which LLVM and SH compilers apply different optimization. Another element that differentiates the estimated and simulated cycles is the number of stalls (e.g., hardware stalls) that is considered only by SH simulator. Furthermore, in SH simulation, there are some cycles of overhead for starting the tasks execution and for synchronization with the host processor; these cycles are ignored by the code analysis tool (as previously mentioned, we are considering these extra cycles during our scheduling analysis to compensate the code analysis tool evaluation).

After these considerations we wanted to identify which elements were influencing our design space exploration and our UM to understand why with such considerable errors in the upper and lower bound estimations, it is still possible to get good results for our case study. We noticed that what is relevant is not the absolute value of the C_j^l and C_j^u of each task, but its relative value when compared to the other tasks in the application

(this is true until a certain extent as there is also the influence of the messages and the communication architecture). Therefore, we evaluated the contribution of each task to the total number of cycles of the application: we performed this check for C_j^l and *sim* values. We used the lower bound value because it corresponds to the most parallelized version of the application and we ran the entire application on an ASIP with a large number of ISs, also to get the most parallelized execution[2]. The results obtained are available in columns 5 and 6 of Table 4.6. Once we had the relative contribution ($\%Tot_{C_j^l}$ and $\%Tot_{sim}$) of each task to the total number of cycles (for the estimated C^l and simulated *sim*), we calculated the absolute error between them. The absolute error is available in column 7 of Table 4.6. According to this evaluation, the estimated upper bound shows which tasks are more time consuming than others and this is reflected also in the simulation results on a real ASIP. We got errors up to $\sim 5\%$ for MJPEG encoder.

To verify how big could be the difference between the estimated and simulated performance values before impacting the DSE results, we increased the upper and lower bounds of each task of 10%, 20%, 35% and 45%. Then we ran our DSE and in all cases we found the same task clustering solution obtained with the original values (with different probability and different quantile values at 50%). This suggests that our DSE is not sensitive to even significant variations in the upper and lower bound estimations.

Table 4.6: Comparison between the number of cycles estimated by the profiling tool [43] and the ones obtained from simulation for MJPEG

Task Name	C_j^l	C_j^u	sim	$\%Tot_{C_j^l}$	$\%Tot_{sim}$	Err $\lvert\%Tot_{C_j^l} - \%Tot_{sim}\rvert$
mainDCT	25,696,592	71,624,910	79,407,360	51.38	55.47	4.09
mainQ	6,402,512	8,875,470	11,895,705	12.80	8.31	4.49
mainVLE	16,335,196	25,787,955	41,572,680	32.66	29.04	3.62
main-VideoOut	1,578,078	1,958,760	3,149,475	3.26	2.20	0.96
Total (cycles)	**50,012,378**	**108,247,095**	**143,166,046**			

4.4 Summary

In this section, we presented the case studies that we used for demonstrating the effectiveness of our DSE with UM. We used both synthetic and real case studies. For two case studies (*Synth 1* and MJPEG encoder), we also implemented the multi-ASIP

[2]The total number of simulated cycles is bigger than the sum of the cycles of the single tasks; these extra cycles are due to the execution of a wrapper function that invokes all the tasks in the task cluster.

platforms corresponding to the task clustering solutions found by our approach. We demonstrated that our DSE is able to find good task clustering solutions, which performances are reflected in simulation after the implementation of the multi-ASIP systems. Furthermore, we analyzed the influence of the upper and lower bound values on the results of the DSE. This analysis suggests that our approach can work properly even in the presence of considerable errors in the upper and lower bound estimations of the UM when compared to the simulated results.

CHAPTER 5

Uncertainty model with SDFG

After the evaluation of the UM using a task graph model, we investigate the impact of the selection of a different application model on our DSE with UM. The task graph is a course grain dataflow model. We decided to use a SDFG that maintains compatibility with the task graph, but offers a lower level of granularity. In fact, a task in a task graph corresponds to its associated task in the SDFG repeated a certain number of times [39]. This allows the SDFG model to exploit more parallelism during scheduling. In Figure 5.1 we provide an intuitive comparison between the scheduling of a task graph (Figure 5.1a) and of a SDFG (Figure 5.1b) for the same application. We assume that τ_1 and τ_2 are mapped to two different PE. With the SDFG model, we can exploit a higher pipeline parallelism between the different PE.

In this chapter we propose a schedulability analysis that uses our UM applied to an application modeled as a SDFG. We use the CP model for representing the platform. The main difference is the introduction of a new algorithm for the schedulability analysis of a task clustering solution. In fact, the SDFG allows exploiting both task level and pipeline parallelism and this is reflected in better performance after the multi-ASIP platform implementation. In particular in Section 5.1 we describe the SDFG that we use for modeling our application. Then, in Section 5.2 we define the schedulability analysis for a single task clustering solution using the SDFG model.

The DSE is implemented with the same SSEA described in Section 3.3.3.

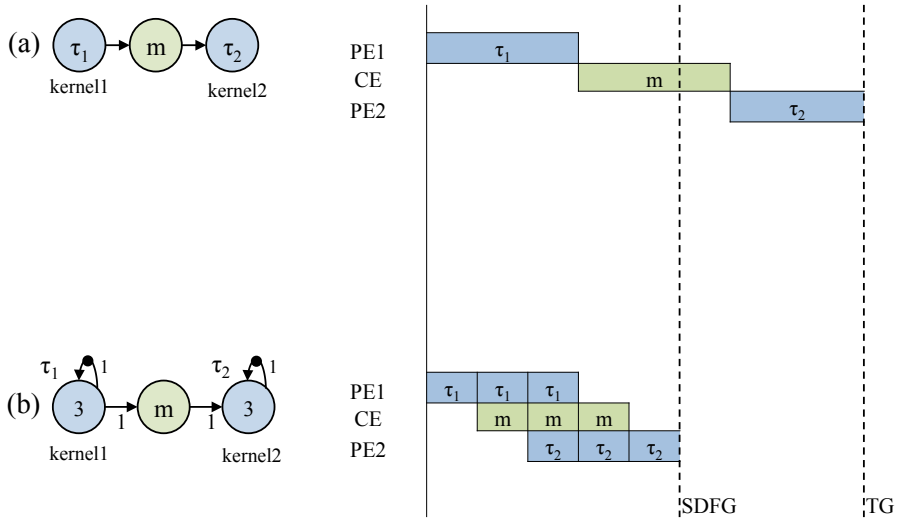


Figure 5.1: Examples of TG (a) and SDFG models (b) of the same application

Part of the material presented in this chapter is also described in [70] that is currently under revision.

5.1 Application model

We assume to be given an application A_i, modeled as a SDFG [58]. A SDFG is defined as a tuple $A_i = (\Gamma_i, \Omega_i, I_i, O_i, Z_i)$ where each element in Γ_i is an *actor* (i.e. a *task*) and each element in Ω_i is an *edge* (i.e. a *message*) that models the communication between actors. Each actor executes by reading *tokens* (i.e. data) from its input messages, and by writing the results of the computation as tokens to the output messages. The number of tokens are called *consumption* and *production rates* and they are contained in the sets I_i and O_i, respectively. In a SDFG, every time an actor executes, it consumes the same amount of tokens from its input messages and produces the same amount of tokens on its output messages. An actor can execute only if the required tokens are available at its input messages. We use the term *firing* to indicate an actor execution. Z_i is the set of the *initial tokens*, i.e. the tokens already available on the messages before the execution of the actors.

Moreover, we consider SDFGs without *auto-concurrency*: we do not allow multiple and simultaneous firings of the same actor. This property can be forced adding a self-

loop to each actor with an initial token [18]. We also consider *consistent* SDFGs [31]: a graph is consistent if we can fire each task a fixed number of times and this will bring the SDFG to its original state, i.e. with the same distribution of tokens over all edges (messages). Each message has associated a value M_g that corresponds to the amount of data transmitted, i.e. the size of a token in bits. An example of SDFG graph for the MJPEG encoder is shown in Figure 5.2, in which the number inside each task indicates the number of firing of that task. In Figure 5.2, to simplify the graphical representation, we represent the self-loop to limit the auto-concurrency, without inserting a message (there is no transfer of data associated). Additionally, we identify *source*, *sink* and *transformer* actors. Source actors have no input messages, while sink actor have no output messages. They do not contain any computation, but they are an interface with the environment. For the MJPEG application for example, we have two source actors that set some environment variables and that simulate the arrival of frames for the elaboration and a sink actor that registers if the application terminated correctly. The tasks that are neither sink nor source are transformer (computation) actors. Moreover, we use as additional information to model the application, the value $iter_i$ that indicates the number of times that we want to execute the SDFG of the application A_i. For example the SDFG in Figure 5.2 models the elaboration of one frame of data. Therefore, a value $iter_{MJPEG} = 15$ expresses the elaboration of 15 frames of data. Each application has a deadline d_i^{iter} that is set according to $iter_i$.

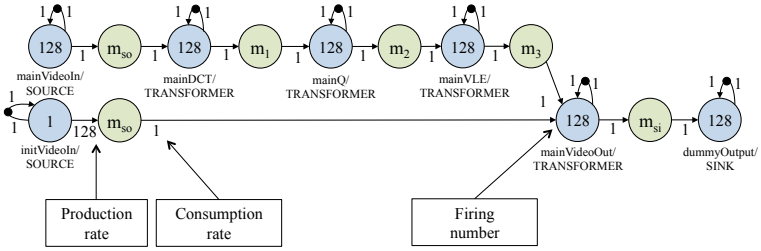


Figure 5.2: SDFG model of MJPEG encoder

5.2 Schedulability Analysis

As presented in the previous chapters, we perform a schedulability analysis to determine the likelihood of a clustering solution to be schedulable once the corresponding multi-ASIP platform is implemented, i.e. we want to determine the probability of having a schedule length δ_{A_i} that meets the deadline d_i^{iter}, $p_i = P(\delta_{A_i} \leq d_i^{iter})$. In our scheduling, we assume that a task is assigned always to the same processor for all its firings. Moreover, if communicating tasks are assigned to the same PE, the message

between them is ignored as the communication time is negligible. As described in Section 3.3.1, we use Monte Carlo simulation (*MCS*). For each task, we extract n random Monte Carlo samples according to the normal distribution modeling the WCET of the task. For each message we have an array that contains n equal values, i.e. the transmission time associated to the message (C_{m_g}). We use the boldface formatting to indicate that a symbol corresponds to a Monte Carlo array of n elements. For the task graph based schedulability analysis (see Section 3.3.1), we use $+$ (sum) and **max** operations. For this analysis, we add $-$ (subtract) and $*$ (multiply) operations. Each operation is performed element by element on arrays of n samples.

In our analysis we consider both task level and pipeline parallelism. This analysis is executed during DSE and, hence, we want to reduce the computation time for the evaluation of a single task clustering solution (as Monte Carlo simulation is already time consuming). Instead of estimating the *schedulability probability* for a SDFG A_i that is repeated $iter_i$ times (as explained in Section 5.1), we perform two separated analysis. First, we run the schedulability analysis of A_i for a single iteration ($iter_i = 1$); we indicate the output of this analysis as δ'_{A_i}. Second, we use δ'_{A_i} and the pipeline properties to estimate the scheduling probability for the total number of iterations (e.g., $iter_i = 5$) that corresponds to δ_{A_i}. We call these two analysis Task-Level Analysis (TLA) and Pipeline Analysis (PA). We use the application in Figure 5.3a as an illustrative example of our schedulability analysis with stochastic variables. For simplicity we assume that each task produces and consumes a single token and that it is fired a single time (in the graphic representation, we omit the self-loops that limit the auto-concurrency). Additionally, we set $iter = 3$. We cluster the application in Figure 5.3a on a two-ASIP platform according to the task clustering solution in Figure 5.3b where tasks are assigned to PE_1 and PE_2 and the messages are assigned to the bus (CE).

5.2.1 Example of Task-level Analysis

TLA corresponds to the schedulability analysis performed in Section 3.3.1 using a task graph. The two schedulability analyses have some similarities. Figure 5.4b shows the TLA for the example in Figure 5.3. We perform a schedulability analysis of a single iteration of the entire SDFG. For each firing of a task or execution of a message we calculate its starting and finishing time. For a task τ_j, we indicate the starting time as $\boldsymbol{t_j^s}$ and the finishing time as $\boldsymbol{t_j^f}$. For a message m_g, we use the symbols $\boldsymbol{t_{m_g}^s}$ and $\boldsymbol{t_{m_g}^f}$. Each starting and finishing time is a stochastic variable and is represented by an array of n samples. The maximum finishing time of all sink tasks is the output δ'_{A_i} of the TLA. As for the analysis done using a task graph, we consider both the data dependencies and the hardware resources (PEs and CE) contention. We use single threaded VLIW processor for the platform implementation, so when we cluster multiple tasks on the same PE, we need to force their sequential execution. Our bus model allows the transmission of one message a time. For this analysis we use the operators **max** and $+$

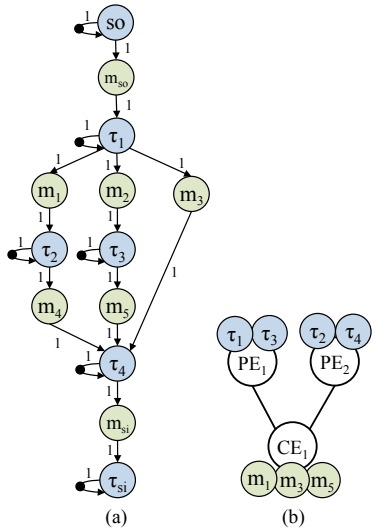


Figure 5.3: Example of application and clustering for δ_{A_i} calculation

(same used in Section 3.3.1).

When it starts, the TLA identifies the tasks that can be fired (i.e. source tasks and *transformer* tasks that have enough tokens at their input edges). For the example in Figure 5.4b, we assume that the WCET of source and sink tasks is equal to zero and we also assume zero bits of data for the m_{so} and m_{si}. Note that the designer can include the source and sink tasks in the schedulability analysis specifying an input option to the DSE tool.

We schedule the source task (τ_{so}) and the output message (m_{so}) and we set their starting and finish times to zeros. Then we schedule τ_1: its starting time $\boldsymbol{t_1^s}$ is given by the maximum of the finishing times (we apply the **max** operator) of all the tasks and messages that τ_1 depends on, in this case only m_{so}. The finishing time t_1^f of τ_1 is given by the sum of the estimated starting time and the n samples extracted by the WCET probability distribution of task τ_1, i.e., $\boldsymbol{t_1^s + C_1}$ ($+$ operation). We can then schedule the remaining tasks following the same rules. All the steps are described in Figure 5.4b. The finishing time of task τ_{si} is a set of n samples that corresponds to the stochastic variable δ'_{A_i}. The algorithms used for TLA are described in Section 5.2.3 (Algorithms 3, 4 and 5).

When multiple tasks and messages are assigned to the same resource and are ready for execution, their scheduling order is determined by our scheduling policy described in Algorithm 5.

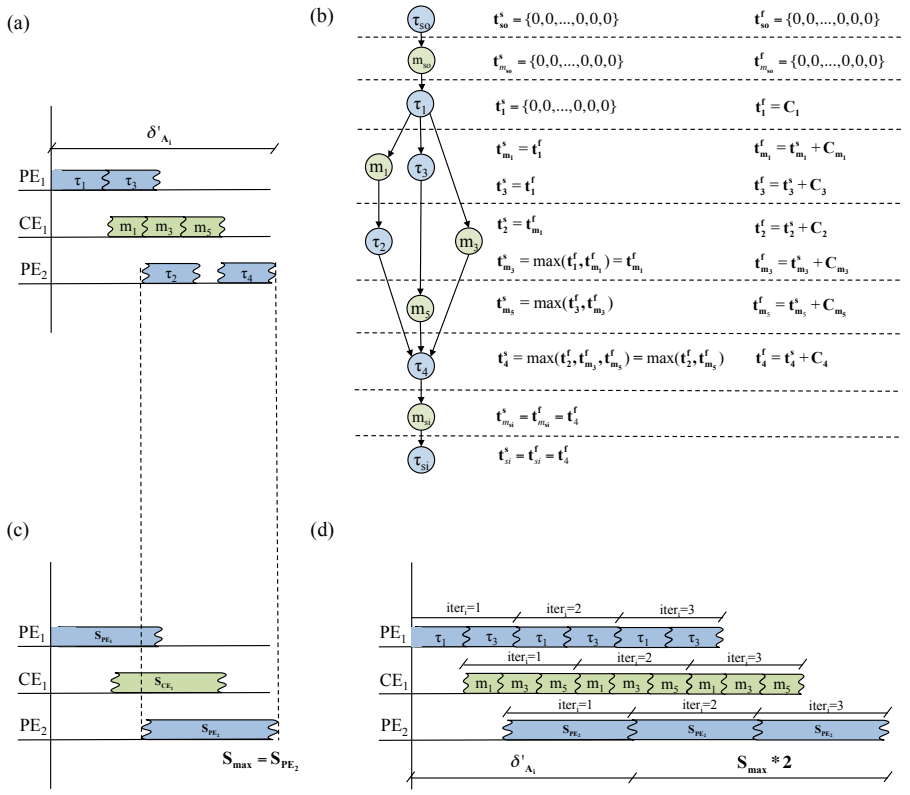


Figure 5.4: Examples of TLA, (a) and (b), PA, (c) and (d), for the SDFG in Figure 5.3

During the TLA, we also collect additional information that is needed by the PA. To estimate the pipeline parallelism at macro-architecture level, we require the size of the pipeline stages. In our case, each pipeline stage S corresponds to an hardware resource, i.e. PEs and CE. We use S_{PE_k} (or S_{CE}) to indicate the size of a pipeline stage: these are also stochastic variables and their probability distribution is modeled by a set of n samples (obtained through MCS). We estimate the size of the pipeline stage $\boldsymbol{S_{PE_k}}$ as described in Equation 5.1, where $\boldsymbol{S^s_{PE_k}}$ and $\boldsymbol{S^f_{PE_k}}$ represent the probability distribution (n-element array) of the first and last time the resource PE_k has been used during the TLA of the application A_i.

$$\boldsymbol{S_{PE_k}} = \boldsymbol{S^f_{PE_k}} - \boldsymbol{S^s_{PE_k}} \tag{5.1}$$

We use the $-$ operator between the MCS n-element arrays. More precisely, $\boldsymbol{S^s_{PE_k}}$ is equal to the starting time of the first task scheduled on PE_k (first firing of the task); $\boldsymbol{S^f_{PE_k}}$ is equal to the finishing time of the last task scheduled on PE_k (last firing of the task). For example, for PE_2 in Figures 5.4a and 5.4c we have $\boldsymbol{S^s_{PE_2}} = \boldsymbol{t^s_2}$ and

$\boldsymbol{S^f_{PE_2}} = \boldsymbol{t^f_4}$. The same approach is used for the messages to evaluate $\boldsymbol{S_{CE}}$.

5.2.2 Example of Pipeline Analysis

In general, the pipelined execution of N elements can be estimated as the sum of the time required by the first element to go through the entire pipeline plus the time required by the remaining $(N-1)$ elements to complete their execution when all pipeline stages are fulfilled. When the stages of the pipeline have different sizes, the time needed for the elaboration of one of the $(N-1)$ elements corresponds to the stage of the pipeline with the maximum size. We use this definition of pipelined execution to define our PA. The time required by the first element to go through the entire pipeline is the δ'_{A_i} produced by the TLA. From TLA we also have the probability distributions modeling the WCET of each pipeline stage ($\boldsymbol{S_{PE_k}}$ or $\boldsymbol{S_{CE}}$). Then we can apply Equation 5.2 to obtain the set of n samples modeling the biggest pipeline stage $\boldsymbol{S_{max}}$ and Equation 5.3 to get $\boldsymbol{\delta_{A_i}}$. In Figure 5.4c and Figure 5.4d, there is an example of the different pipeline stages and of the computation of the $\boldsymbol{\delta_{A_i}}$.

$$\boldsymbol{S_{max} = max(S_{PE_1}, ..., S_{PE_{PC_{max}}}, S_{CE})} \tag{5.2}$$

$$\boldsymbol{\delta_{A_i} = \delta'_{A_i} + S_{max} * (iter_i - 1)} \tag{5.3}$$

The PA can be applied to speed up the schedulability analysis **only if there are no data dependencies between successive iterations of the SDFG**. In case of dependencies, it is possible to use TLA to compute the schedulability analysis for all the iterations of the SDFG; the main disadvantage is a higher computation time to perform the schedulability analysis.

5.2.3 Algorithms for schedulability analysis

In this section we describe the pseudo-code for TLA and PA given an application A_i, a clustering solution Sol_c and a platform constraint PC_{max}. Algorithm 2 lists the steps for performing TLA (from line 1 to 10) and PA (from line 11 to 13) analyses as presented in Sections 5.2.1 and 5.2.2. The variable $activeTaskList$ contains the list of tasks that can be scheduled as their data dependencies are satisfied (source tasks or tasks with enough tokens on their input messages). At each iteration of the algorithm ($while$ loop at line 1), we update $activeTaskList$; the head element of the list is the next task to be scheduled. Algorithm 3 describes the steps for scheduling a single task τ_j. We need to determine the starting time for τ_j (lines 1 to 4). A task can start:

- as soon as the tasks (and messages) from which it depends on are completed
- as soon as the hardware resource to which the task is assigned is available

Then we can calculate the finishing time of τ_j, update the number of token consumed (lines 10 to 12) and produced (lines 13 and 14) and schedule the messages in output of τ_j (lines 15 and 16). Every time we schedule a task τ_j, we also schedule its output messages if they are connecting τ_j with a task in a different cluster; we assume that each task reads data from local memory and writes them to remote memory. For example, let us assume two communicating tasks τ_1 and τ_2 assigned to two different processors PE_1 and PE_2. In our schedulability analysis, τ_1 reads the input data from the local memory of PE_1 and writes the output data to the memory of PE_2 (Figure 5.5 shows an example). Once the scheduling of a task is completed, we need to update the $activeTaskList$ as specified in Algorithm 4. Algorithm 3 is also used to calculate the starting (lines 6-8) and finishing time (lines 17-20) of each pipeline stage (PEs and CE).

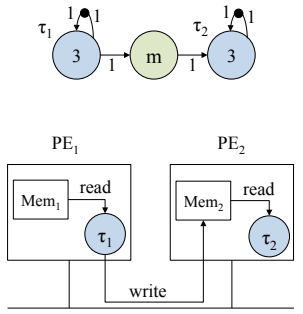


Figure 5.5: Reading and writing policy: τ_1 reads from local memory and writes to remote memory

Finally, Algorithm 5 is used to determine which is the next task in $activeTaskList$ that can be scheduled. The purpose of Algorithm 5 is to produce a scheduling where we fire all tasks maintaining the smallest relative distance between the actual number of firings of each task (while building the scheduling) and the number of firings in the original SDFG. We interleave the execution of the tasks and avoid consecutive firings of the same task. For each message we calculate the rate between the number of firings of the target task (task consumer) and the number of firings of the source task (task producer) and we call this $idealRate$. At each iteration of the algorithm, we calculate, for each message, the $realRate$ that is the rate between the actual number of times the task consumer and producer have been fired. When the algorithm starts, we initialize the $realRate$ variable to ∞. Then we estimate the relative error ($distanceRate$) between the $realRate$ and the $idealRate$ of each message and we sort the messages in descending order according to their relative errors. The sorted order of the messages

determines the firing order for the tasks. If the $realRate$ for a message m_g is bigger than the $idealRate$, then we should fire the task producer associated to m_g, otherwise the task consumer (the tasks must be available in the $activeTaskList$).

In Figure 5.6c there is an example of eight iterations of Algorithm 5 for the application in Figure 5.6a. We reported the steps for the scheduling of one iteration of the SDFG (Figure 5.6a). Figure 5.6b contains the values of the $idealRate$ variable for each message. At each iteration, we fire τ_{next} and we update the values of the $realRate$ and $distanceRate$. Then we consider one message at a time, starting with the one with the highest $distanceRate$ (when multiple messages m_g have the same $distanceRate$, we prioritize them according to their identifier g). For the selected message, we verify if we should fire its source or target task; for clarity, in Figure 5.6c (columns 15-18), we reported the task that we should fire according to the $realRate$ and $idealRate$ of each message. The selected source/target task must be available in the $activeTaskList$, otherwise we select the message with the next highest $distanceRate$ until it is possible to assign the variable τ_{next}. If no task can be assigned according to this policy, we assign to τ_{next}, the last task that has been added to $activeTaskList$. As an example, let us consider the row corresponding to $Time = 4$ in Figure 5.6c: we calculate the $realRate$ and $differenceRate$ for each message. Then, we sort the messages according to their $differenceRate$ in descending order, obtaining the following sorted list: m_1,m_4,m_2 and m_3. We consider the first message in this sorted list: m_1. Afterwards, we determine if we should fire the source or target task of m_1 (columns 15 in Figure 5.6c). As the $realRate$ of m_1 is smaller than its $idealRate$ (0 and 0.5, respectively), we select the target task (i.e. τ_2). As the last step, we verify if τ_2 is in the $activeTaskList$ (columns 14); if this is the case, we can set $\tau_{next} = \tau_2$, otherwise, we proceed with the next message in the sorted list (i.e., m_4), until we find a task that can be fired. Using Algorithm 5, we guarantee that during the schedulability analysis we fire the tasks in a order that can be easily reproduced on the SH ASIPs.

5.3 Comparison of clustering solutions and DSE

For the comparison of task clustering solutions, we use the same approach applied for the task graph application model in Section 3.3.2. In a similar manner, for DSE, we use the SSEA implemented for the task graph application model (Section 3.3.3).

Algorithm 2 - Schedulability analysis, $P(\delta_{A_i} \leq d_i^{iter})$ for Sol_c

1: $activeTaskList :=$ source tasks in A_i
2: **while** ($activeTaskList\ NOT\ EMPTY$) **do**
3: $\tau_j := activeTaskList.head$
4: $\{\boldsymbol{t_j^f}, \boldsymbol{S_{PE_k}^s}, \boldsymbol{S_{PE_k}^f}, \boldsymbol{S_{CE}^s}, \boldsymbol{S_{CE}^f}\}$:= Schedule τ_j (Algorithm 3)
5: Update $activeTaskList$ (Algorithm 4)
6: Select from $activeTaskList$ the next task τ_{next} to be scheduled (Algorithm 5)
7: Set τ_{next} as $activeTaskList.head$
8: **end while**
9: $\boldsymbol{\delta'_{A_i}}$:= Find the maximum $\boldsymbol{t_j^f}$ of the sink tasks
10: $\{\boldsymbol{S_{PE_1}}, ..., \boldsymbol{S_{PE_{PC_{max}}}}\}$:= Calculate the size of the pipeline stages using $\boldsymbol{S_{PE_k}^s}$, $\boldsymbol{S_{PE_k}^f}$, $\boldsymbol{S_{CE}^s}$ and $\boldsymbol{S_{CE}^f}$
11: $\boldsymbol{S_{max}}$:= Find the maximum in $\{\boldsymbol{S_{PE_1}}, ..., \boldsymbol{S_{PE_{PC_{max}}}}, \boldsymbol{S_{CE}}\}$
12: $\boldsymbol{\delta_{A_i}}$:= Calculate $\boldsymbol{S_{max}} * (iter_i - 1) + \boldsymbol{\delta'_{A_i}}$
13: Calculate $p_i = P(\delta_{A_i} \leq d_i^{iter})$

Algorithm 3 - Schedule τ_j

1: $dataDependencyList :=$ Find data dependencies of τ_j
2: $\boldsymbol{S_{PE_k}^f}$:= Find last time the resource PE_k has been used (τ_j assigned to PE_k)
3: $\boldsymbol{t_{dep}^f}$:= Find the maximum of $\boldsymbol{t_{j1}^f}$ for each τ_{j1} in $dataDependencyList$
4: $\boldsymbol{t_j^s}$:= Find the maximum between $\boldsymbol{t_{dep}^s}$ and $\boldsymbol{S_{PE_k}^f}$
5: $\boldsymbol{t_j^f}$:= Calculate the finishing time ($\boldsymbol{t_j^s + C_j}$)
6: **if** first time PE_k used **then**
7: $\boldsymbol{S_{PE_k}^s} := \boldsymbol{t_j^s}$
8: **end if**
9: $\boldsymbol{S_{PE_k}^f} := \boldsymbol{t_j^f}$
10: **for** EACH of the input messages m_g of τ_j **do**
11: Update the number of token available on m_g (token consumed)
12: **end for**
13: **for** EACH of the output messages m_g of τ_j **do**
14: Update the number of token available on m_g (token produced)
15: **if** m_g connect τ_j to a task in a different cluster **then**
16: Schedule m_g on the bus CE
17: **if** first time CE used **then**
18: $\boldsymbol{S_{CE}^s} := \boldsymbol{t_{m_g}^s}$
19: **end if**
20: $\boldsymbol{S_{CE}^f} := \boldsymbol{t_{m_g}^f}$
21: **end if**
22: **end for**

Algorithm 4 - Update $activeTaskList$

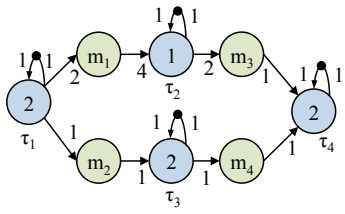
```
1: for EACH of the input messages m_g of τ_j do
2:     if NOT enough token in input then
3:         Remove τ_j from activeTaskList
4:     end if
5: end for
6: for EACH of the output messages m_g of τ_j do
7:     Identify target task τ_j1 (connected to m_g)
8:     for ALL input messages m_g1 of τ_j1 do
9:         if enough token in input then
10:            Add τ_j1 to activeTaskList
11:        end if
12:    end for
13: end for
```

Algorithm 5 - Select from $activeTaskList$, the next task τ_{next} to be scheduled

```
1: if First invocation of Algorithm 5 then
2:     for EACH of the messages m_g of A_i do
3:         idealRate[m_g] := (Total number of firing of target task of m_g)/(Total number of firing
           of source task of m_g)
4:     end for
5: end if
6: for EACH of the messages m_g of A_i do
7:     realRate[m_g] := (Actual number of firing of target task of m_g)/(Actual number of firing
       of source task of m_g)
8:     distanceRate[m_g] := Calculate relative error between realRate[m_g] and idealRate[m_g]
9: end for
10: sortedMessagesList := Sort messages in descending order based on the distanceRate
11: for EACH of the messages m_g in sortedMessagesList do
12:    if realRate[m_g] > idealRate[m_g] then
13:        τ_temp := source task of m_g
14:    else
15:        τ_temp := target task of m_g
16:    end if
17:    if τ_temp is in activeTaskList then
18:        τ_next = τ_temp
19:        return
20:    end if
21: end for
22: τ_next := last task that added to activeTaskList
```

5.4 Summary

In this chapter we introduced the SDFG as new application model for our DSE with UM. A SDFG allows exploiting both task level and pipeline parallelism, as each task



(a)

idealRate			
m_1	m_2	m_3	m_4
0.5	1	2	1

(b)

Time	#firing				realRate				distanceRate (1-(Real rate/Ideal rate))				activeTask List	(realRate > idealRate)? source:target				Next task τ_{next}
	τ_1	τ_2	τ_3	τ_4	m_1	m_2	m_3	m_4	m_1	m_2	m_3	m_4		m_1	m_2	m_3	m_4	
1	0	0	0	0	∞	∞	∞	∞	-∞	-∞	-∞	-∞	τ_1	τ_1	τ_1	τ_2	τ_3	τ_1
2	1	0	0	0	0.0	0.0	∞	∞	1	1	-∞	-∞	τ_1, τ_3	τ_2	τ_3	τ_2	τ_3	τ_3
3	1	0	1	0	0.0	1.0	∞	0.0	1	0	-∞	1	τ_1	τ_2	τ_1	τ_2	τ_4	τ_1
4	2	0	1	0	0.0	0.5	∞	0.0	1	0.5	-∞	1	τ_2, τ_3	τ_2	τ_3	τ_2	τ_4	τ_2
5	2	1	1	0	0.5	0.5	0.0	0.0	0	0.5	1	1	τ_3, τ_4	τ_2	τ_3	τ_4	τ_4	τ_4
6	2	1	1	1	0.5	0.5	1.0	1.0	0	0.5	0.5	0	τ_3	τ_2	τ_3	τ_2	τ_3	τ_3
7	2	1	2	1	0.5	1.0	1.0	0.5	0	0	0.5	0.5	τ_4	τ_2	τ_1	τ_2	τ_4	τ_4
8	2	1	2	2	0.5	1.0	2.0	1.0	0	0	0	0	-					

(c)

Figure 5.6: (a) Input SDFG, (b) idealRate for each message of the SDFG, (c) results of the first eight iteration of Algorithm 5

has a smaller level of granularity than the corresponding task in a task graph. We used the same evolutionary algorithm (SSEA) with MCS as we did for the task graph application model. The main difference is the introduction of a new schedulability analysis that is divided into two parts, TLA and PA. TLA represents the core of our schedulability analysis, while PA has been introduced mainly to speed up the analysis when we consider multiple iteration of the same SDFG.

In the next chapter, we will use a real case study to compare the schedulability analysis obtained with a task graph and a SDFG application models. Additionally, we will present multiple real case studies modeled as a SDFG and with a large number of tasks and iterations, to which we will apply our DSE with UM to prove the effectiveness of our analysis.

CHAPTER 6

Experimental evaluation with the SDFG model

This chapter contains a number of case studies that demonstrate the effectiveness of our DSE with UM when the application model is a SDFG. Firts, we use the MJPEG encoder (already used in Section 4.2) as a SDFG and we apply our DSE; as we did for the application modeled as a task graph, we compare different task clusterings to demonstrate the effectiveness our schedulability analysis using a SDFG and the UM (described in Section 5.2). We use SH tools to implement the multi-ASIP platforms and validate the results obtained. Moreover, we compare the results obtained using the task graph with the ones obtained using the SDFG as application model.

We considered two additional streaming applications, taken from multimedia and medical domains and that fit well with the SDFG model: the spatial coding (SC) algorithm extracted from the MJPEG4 application (property code of STMicroelectronics [91]) and the Electrocardiogram (ECG) application [93]. We apply our DSE and we validate the results using SH tools. We also analyze of the sensitivity of our approach to the accuracy of the WCET bounds.

In addition, at the end of the section, we present a small example in which the bus is substituted by a NoC. This example is used as a proof of concept to demonstrate that our probabilistic schedulability analysis can be applied to communication architecture more complex than a bus.

This chapter is organized as follows. Section 6.1 describes the DSE applied to the MJPEG application, while Section 6.2 briefly describes the results obtained for the ECG and SC case studies. Section 6.3 contains a comparison of the task graph and of the SDFG models using the MJPEG encoder as an example. Section 6.4 analyzes the influence of the upper and lower bounds of the UM on the DSE when using a SDFG model; for the MJPEG encoder application, it also presents the outcome of the DSE when we use more precise WCETs to build our UM. Finally Section 6.6 presents our UM applied to the MJPEG case study with a NoC as initial platform.

Part of the results presented in this chapter are also available in [70] that is currently under revision.

6.1 Case study: MJPEG encoder

In this section we describe the results obtained after running our DSE with UM for the MJPEG encoder. We model the application as a SDFG and we use the schedulability analysis (TLA and PA) described in Section 5.2.

We used the same design flow applied to the MJPEG encoder modeled as a task graph that is described in Figure 4.1 with the support of the same external tools: Compaan [1] for the application partitioning, Phase 1 and 2 of ASAM micro-architecture DSE [43] for deriving the upper and lower bounds and for the definition of the ASIP micro-architecture and SH tools [55] for the development and simulation of the multi-ASIP platform.

In the first row of Table 6.1, there are the input constraints for the MJPEG encoder application: we considered the elaboration of 15 frames and a desired throughput of 30 frame-per-second (fps) that give a deadline d^{15}_{MJPEG} of $0.5s$. We used a CP model with $PC_{max} = 3$ and a frequency $f = 166MHz$. As input data, we used the ones employed for the case study with the task graph model and with the highest number of cycles estimated by the code analysis tool.

Table 6.1: Input constraints for MJPEG encoder

Case study	$d\ (\mu s)$	$f\ (MHz)$	PC_{max}	*Bus type*
MJPEG encoder	500,000	166	3	b^{166}_{32}

To obtain a SDFG of the application, we edited its C code; we modified the code of each task in the task graph model to get a new task with a lower level of granularity. In Figure 6.1, there is an example that shows the same task as part of a task graph and

as part of a SDFG. Then we ran Compaan compiler and from the KPN generated, we extracted the SDFG of the MJPEG encoder (Figure 6.2). We transformed each KPN actor into a task with the notation of the number of firings and each edge into a message annotated with the bits of data to transfer.

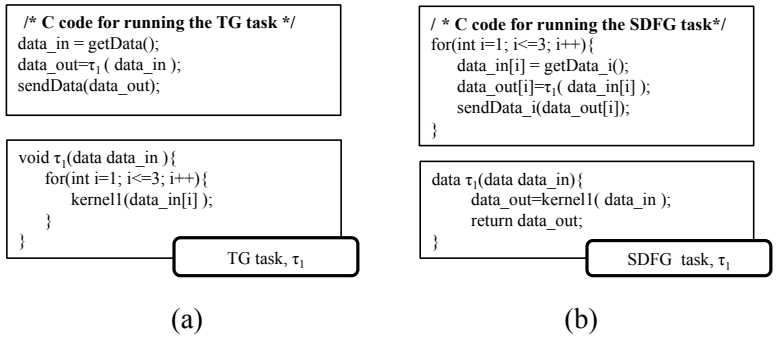


Figure 6.1: Example of C code for a task part of a task graph (a) or of a SDFG (b)

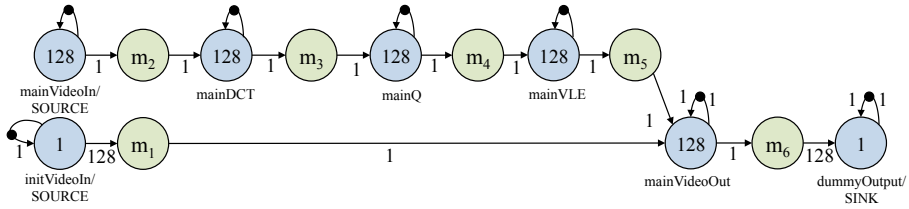


Figure 6.2: SDFG model for the MJPEG encoder

Then we used the code analysis tool (Phase 1 of ASAM micro-architecture DSE [43]) to get the upper and lower bounds (C^l and C^u) for each task: the numbers of cycles associated with each task firing are summarized in Table 6.2. As mentioned in Section 5.1, the source and sink actors of each applications are used for data initialization (i.e. for writing the input data into a local or external memory of the multi-ASIP platform that we want to design), and for providing feedback to the user about the exit status of the application. For this reason, we consider their WCETs equal to zero: they are not assigned to any ASIPs, but their execution is demanded to the host processor. Then we calculated the mean and variance using the estimated C^l and C^u of the tasks and we draw the CDFs for an input frequency $f = 166MHz$ (Figure 6.3). The amount of data associated with each message is shown in Table 6.3. We calculated the transmission time of the messages C_{m_g} using a bus defined as b^f_{32} (i.e. a 32 bit width bus with the same frequency f of the ASIPs), which is compatible with the final implementation of the ASIPs and platform that we have available.

Then we executed our macro-architecture DSE with UM and the schedulability analysis

described in 5.2. We ran the SSEA for 300 s and we used $n = 5,000$ for the Monte Carlo simulation.

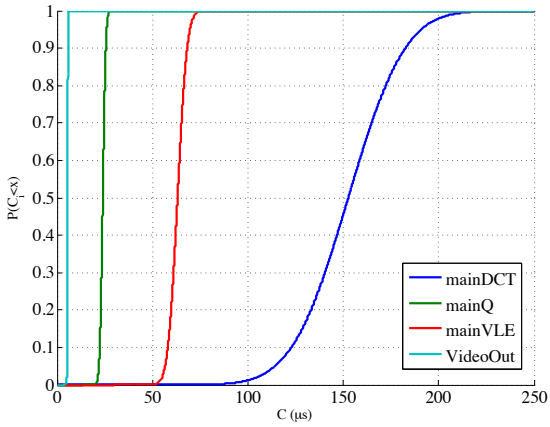


Figure 6.3: Cumulative distribution functions for the tasks of the MJPEG encoder application (with $f = 166MHz$)

Table 6.2: C values for MJPEG encoder (average number of cycles for a single iteration of the task)

C	$mainDCT$	$mainQ$	$mainVLE$	$mainVideoOut$
C_j^u	37301	4619	12513	1008
C_j^l	13383	3334	8445	816

Table 6.3: Message sizes (in bits) for MJPEG encoder

m_1	m_2	m_3	m_4	m_5	m_6
128	8192	8192	8192	4096	32

Figure 6.4 depicts the $P(\delta_{A_{MJPEG}} < d^{15}_{MJPEG})$ produced by our DSE. We found a solution, Sol_1, that has $p_{MJPEG} \sim 1$ to meet the deadline and that uses two ASIPs. The first row of Table 6.4 summarizes the outcome of our exploration: the clustering solution (columns 2-4), the probability of the application to meet the deadline ($P(\delta_{A_{MJPEG}} < d^{15}_{MJPEG})$) and the quantile function value at a probability of 0.5 ($\delta^{0.5}_{A_{MJPEG}} = P^{-1}(p_{0.5})$).

Phase 2 of the micro-architecture DSE tool [43] is then used to get a description of the micro-architecture of the two-ASIPs. For the two task clusters found by our DSE,

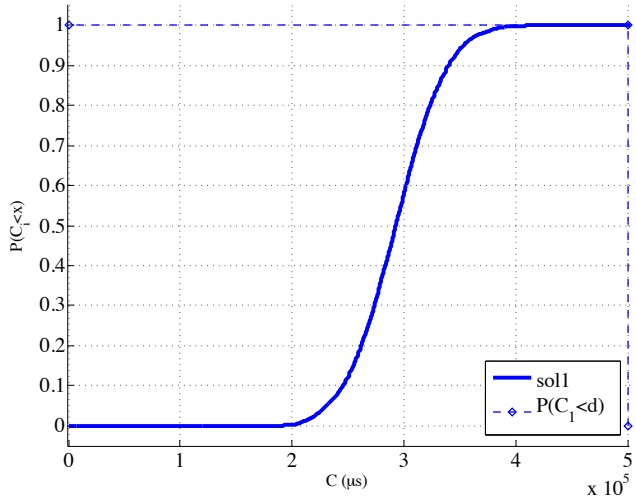


Figure 6.4: Results from the macro-architecture DSE for the MJPEG encoder

Phase 2 defined two ASIPs, each of them with 3 ISs. After obtaining the micro-architecture description of the ASIPs, we used SH tools; we implemented the two-ASIP platform (Figure 6.6) and we mapped the application code to the ASIPs. In columns 5-6 of Table 6.4 there are the number of execution cycles obtained from SH simulator (sim) and the corresponding time in μs (at a frequency $f = 166MHz$). The task clustering solution found with our DSE meets the deadline $d^{15}_{MJPEG} = 0.5s$ and is, therefore, schedulable ($sched$ column in Table 6.4). Similarly to the analysis done in Section 4.2, our DSE does not provide schedulability guarantees, but it works by comparison: it allows evaluating a task clustering solution against the others and identifies the solution with highest chances of producing a schedulable implementation after the platform is available.

To prove that our analysis is effective, we considered other task clustering solutions than the one found by the DSE: we evaluated each solution with our schedulability analysis and UM and with SH tools. The results obtained are in Figure 6.5 and in Table 6.4. We tested the same task clustering solutions evaluated with the task graph application model. We added an additional solution with three processors (Sol_6), to further exploit the pipeline parallelism that can derive from using an additional ASIP. When multiple solutions have the same probability, we prefer the one with the smaller number of clusters and the smaller quantile value $P^{-1}(p_{0.5})$. The results in Table 6.4 shows that our DSE using a SDFG as application model is able to determine which solution is better than the other. Sorting the task clustering solutions (from the best to the worst one) according to our schedulability analysis, we find that their order

matches the results obtained with the cycle-accurate simulator from SH. This shows that our schedulability analysis with UM is able to properly evaluate the different task clustering solutions and find the ones that are more promising for platform synthesis, supporting the designer and speeding up the design process.

The last column in Table 6.4 shows that only Sol_1 is schedulable (at a frequency $f = 166MHz$). $mainDCT$ is the task with the highest number of cycles, and, therefore, it is better to have a dedicated ASIP for its execution. Sol_4 and Sol_6 also have a dedicated processor for $mainDCT$, but they have a slightly higher quantile value for the additional communication time due to the exchange of data among three processors. This is verified with both the UM and the SH simulation. Moreover, from the results in Table 6.4, we can observe that there is a relation between the quantile value of each solution and the simulated execution time for most of the task clustering solutions. For example let us considered the slowest and the fastest task clustering solutions: Sol_1 and Sol_5. The rate between the quantile values of the two solution is $292,700/470,080 = 0.622$ and the rate between the simulated cycles (sim) $79,088,561/126,635428 = 0.624$. This suggests that the results found by our DSE with UM are consistent with the one obtained after the implementation of the platform.

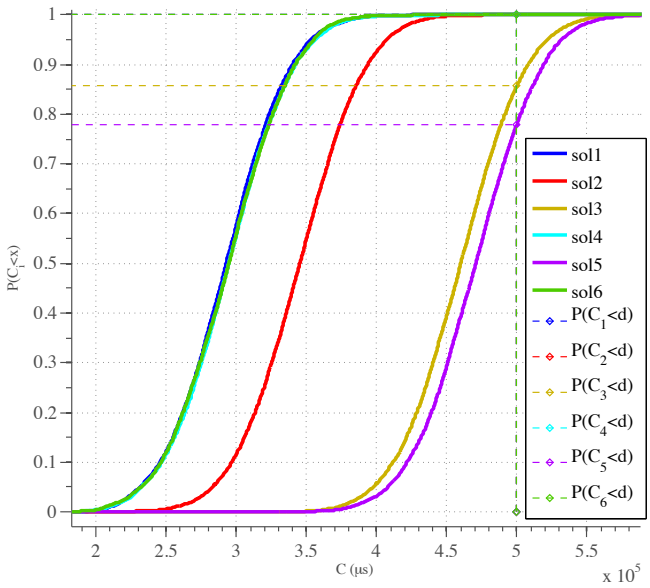


Figure 6.5: Comparison of the CDF of different clustering solutions for the MJPEG encoder

Table 6.4: Comparison of clustering solutions for MJPEG encoder

Sol_{ID}	Clusters			$P(\delta_{A_{MJPEG}} < d^{15}_{MJPEG})$	$\delta^{0.5}_{A_{MJPEG}} = P^{-1}(p_{0.5})$ (μs)	*sim* (cycles)	*sim* (μs)	sched
	PE_1	PE_2	PE_3					
1	mainDCT	mainQ, mainVLE, mainVideoOut	-	~ 1	292700	79088561	476437.11	yes
2	mainDCT, mainQ	mainVLE, mainVideoOut	-	~ 1	345300	85740371	516508.26	no
3	mainDCT, mainQ, mainVLE	mainVideoOut	-	0.85	460100	124194971	748162.48	no
4	mainDCT	mainQ, mainVLE	mainVideoOut	~ 1	294700	83617556	503720.22	no
5	mainDCT, mainQ, mainVLE, mainVideoOut	-	-	0.77	470800	126635428	762864.02	no
6	mainDCT	mainQ	mainVLE, mainVideoOut	~ 1	294700	83636411	503833.80	no

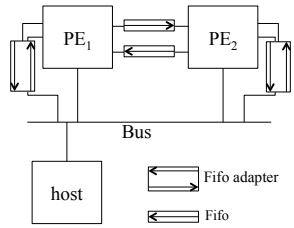


Figure 6.6: Block schematic of the platform generated for Sol_1 for MJPEG encoder

6.2 Case studies: ECG and SC

For ECG and SC, we followed the same design steps described for the MJPEG encoder in Section 6.1. The results obtained for these case studies confirmed the ones obtained for the MJPEG encoder. Therefore, in this section, we briefly present them and their details are available in Appendix A (Sections A.2.1 and A.2.2).

The ECG application contains a cascade of 6 filters (i.e., tasks). The SDFG iterates 10,000 times; therefore, it is a good candidate for our TLA and PA described in Section 5.2. The SC application shows our approach applied to a much wider application compared to the MJPEG encoder: its SDFG contains 27 transformer tasks and it iterates 6,000 times. For ECG we have a deadline of $16s$ considering the elaboration of 10,000 samples and for SC, a deadline of $0.205s$ for the elaboration of five frames. We set $PC_{max} = 2$ and $PC_{max} = 4$, respectively.

The results obtained for ECG and SC are summarized in Tables 6.5 and 6.6, respectively. In both tables, Sol_1 is the solution produced by our DSE with UM, while the other solutions are used for comparison. We select those solutions that are reasonable alternatives to the task clustering produced by the DSE. We also evaluate the solutions with a single task cluster (Sol_5 in Table 6.5 and Sol_2 in Table 6.6). For both ECG and SC there is a correspondence between the results obtained from our DSE and the ones returned by the simulation with SH tools after the platform implementation. This means that we can use our UM to compare different task clustering solutions. The UM, taking into consideration a range of WCETs, is able to find a valid task clustering solution and guide the designer during the implementation of a multi-ASIP platform.

Table 6.5: Comparison of clustering solutions for ECG

Sol_{ID}	Clusters		$P(\delta_{A_{ECG}} < d_{ECG}^{10,000})$	$\delta_{A_{ECG}}^{0.5} = P^{-1}(p_{0.5})$ (μs)	*sim* (cycles)	*sim* (μs)	sched
	PE_1	PE_2					
1	lowpass, highpass, derivative, square	integral, detect	0.56	15783000	13790796	13790796	yes
2	lowpass, highpass, derivative	square, integral, detect	0.54	15853000	14000776	14000776	yes
3	lowpass, highpass	derivative, square, integral, detect	0.52	15928000	14460733	14460733	yes
4	lowpass, highpass, derivative, square, integral	detect	0.23	16942800	15934010	15934010	yes
5	lowpass, highpass, derivative, square, integral, detect	-	0.22	16991000	16692594	16692594	no

6.3 Comparison of SDFG and task graph application models

The SDFG model allows exploiting an increased level of pipeline parallelism between the multiple firing of the tasks when compared to the task graph model. This becomes evident in the schedulability analysis done with our UM and it is reflected into the performance obtained by the execution of the application on the multi-ASIP platform. For comparing the two application models we need to evaluate the same task clustering solutions using the same input constraints: we used the MJPEG encoder and the task clustering solutions, Sol_1 to Sol_5 specified in Table 6.7. We used a frequency $f = 166MHz$ and a deadline $d = 0.5s$ (in Section 4.2, for the MJPEG encoder application we used a deadline $d = 0.6s$). In Table 6.7, we can observe that for all solutions the schedulability analysis with SDFG returned better performances, i.e. higher probability of meeting the deadline and/or smaller quantile values (columns 5-6 and 8-9). This is reflected also in the performance obtained from simulation with SH technology

Table 6.6: Comparison of clustering solutions for SC

Sol_{ID}	Clusters			$P(\delta_{A_{SC}} < d_{SC}^{6000})$	$\delta_{A_{SC}}^{0.5} = P^{-1}(p_{0.5})\ (\mu s)$	*sim* (cycles)	*sim* (μs)	sched
	PE_1	PE_2	PE_3					
1	MBGetLine1, DCT_{1,2}, MBZero{0,1,2,3,4,5}, MBPackGetLine{1,2}, keep2x2	MMTC_fquantSR	iquantizeSR, MBPackGetLine{3,4}, slRow3, fxIDCT8_{1,2,3,4}, fefoIDCT8_{1,2}, srTrim, srAddRow3, MBPack6, MBPack3	0.99	172000	274847324	171799.58	yes
2	MBGetLine1, DCT_{1,2}, MBZero{0,1,2,3,4,5}, MBPackGetLine{1,2}, keep2x2, MMTC_fquantSR, iquantizeSR, MBPackGetLine{3,4}, slRow3, fxIDCT8_{1,2,3,4}, fefoIDCT8_{1,2}, srTrim, srAddRow3, MBPack6, MBPack3	-	-	0	242600	573715348	358572.09	no
3	MBGetLine1, DCT_{1,2}, MBZero{0,1,2,3,5}, MBPackGetLine{1,2}, keep2x2	MBZero4, MMTC_fquantSR	iquantizeSR, MBPackGetLine{3,4}, slRow3, fxIDCT8_{1,2,3,4}, fefoIDCT8_{1,2}, srTrim, srAddRow3, MBPack6, MBPack3	0.99	185400	275255614	172034.76	yes
4	MBGetLine1, DCT_{1,2}, MBZero{0,1,2,3,4,5}, MBPackGetLine{1,2}, keep2x2, iquantizeSR	MMTC_fquantSR	MBPackGetLine{3,4}, slRow3, fxIDCT8_{1,2,3,4}, fefoIDCT8_1,2, srTrim, srAddRow3, MBPack6, MBPack3	0.99	185500	278939568	174337.23	yes
5	MBGetLine1, DCT_{1,2}, MBZero{0,1,2,3,4,5}, MBPackGetLine{1,2}, keep2x2	MMTC_fquantSR, iquantizeSR, MBPackGetLine3, slRow3, fxIDCT8_1	MBPackGetLine4, slRow3, fxIDCT8_{2,3,4}, fefoIDCT8_{1,2}, srTrim, srAddRow3, MBPack6, MBPack3	0.95	194300	303402927	189626.83	yes

(columns 7 and 10).

An exception is Sol_5 in which we evaluated a task clustering solutions with a single ASIP. With our UM and schedulability analysis, we predicted similar performances for the task graph and SDFG models. This is reasonable as we cannot exploit any pipeline parallelism at system level with a single processor. However, we have a higher number of execution cycles, after implementation, for the task graph model. This is probably due to a different level of optimization performed by SH compiler to the application code. Additionally, the frames of data, to be processed, need to be transferred to the ASIP local memory by the host processor. Depending on the application model, the host sends one frame at a time (in the task graph model) or one block of data extracted from a frame (in the SDFG model). In a frame there are 128 blocks of data. The host cannot send a new frame/block until the previous one has been completely read, otherwise the data would be overwritten and this would produce a wrong output. The host waits until it receives a notification that the memory of the ASIP can be overwritten (we use FIFOs for the synchronization). Therefore, the bigger size of the data in the task graph model implies that the ASIP has to wait a longer time before the data is available. It also needs to elaborate the entire frame before sending the request for new data to the host processor.

It is possible to mitigate this delay using contiguous block of memory and save multiple frames or blocks as in a *buffer*. Let us suppose the case of a *double buffer*: with the task graph model, the host processor sends a frame of data and saves it to the memory location with address $0x4$; the ASIP reads it and at the same time sends the notification to the host that sends the second frame of data to the memory location with address $0x4 + sizeof(frame)$. The third frame is stored again to the address $0x4$. For the experimental evaluation, in the implemented platforms, we used a *double buffer*.

Starting from this discussion, we can also consider the different memory sizes required by the SDFG and task graph application models. Optimizing the local memories of the ASIPs is out of the scope of this thesis, but it can be considered for future work. The task graph model requires the exchange of bigger amount of data compared to SDFG model. The message sizes for the task graph and SDFG are summarized in Tables 4.4 and 6.3, respectively. This means that the local memory of the ASIP, with the task graph model, needs to contain the entire message (128 blocks of data that compose a frame multiplied by the buffer size) and also the intermediate results of the elaboration of the different tasks. For the SDFG model, a smaller amount of memory is required.

From the computation point of view, the SDFG schedulability is more complex (for MJPEG encoder, we ran the DSE with SDFG for 300s, against the 200s of the DSE with the task graph model). For the SDFG, we need to run the evolutionary algorithm for longer time to evaluate a reasonable number of generations, as the evaluation of a single task clustering solution is more time consuming (each task in the SDFG is fired multiple times, e.g. *mainDCT* is fired 128 times).

Table 6.7: Comparison of clustering solutions for MJPEG encoder using SDFG and task graph (TG) application models

Sol_{ID}	Clusters			$P(\delta_{A_{MJPEG}} < d_{MJPEG})$		$P^{-1}(p_{0.5})$ (μs)		sim (μs)	
	PE_1	PE_2	PE_3	TG	SDFG	TG	SDFG	TG	SDFG
1	mainDCT	mainQ, mainVLE, main-VideoOut	-	~ 1	~ 1	311200	292700	547683.62	476437.11
2	mainDCT, mainQ	mainVLE, main-VideoOut	-	0.99	~ 1	354700	345300	624005.51	516508.26
3	mainDCT, mainQ, mainVLE	mainVideoOut	-	0.82	0.85	464200	460100	859913.53	748162.48
4	mainDCT	mainQ, mainVLE	mainVideoOut	~ 1	~ 1	311800	294700	549077.82	503720.22
5	mainDCT, mainQ, mainVLE, main-VideoOut	-	-	0.77	0.77	471500	470800	862446.06	762864.02

6.4 Accuracy of C_j^l and C_j^u

In this section we repeated, for the SDFG application model, the analysis done in Section 4.3, to investigate the influence of the upper and lower bounds (C_j^u and C_j^l) of each task τ_j.

For this analysis, we considered the MJPEG encoder. First, we evaluated the accuracy of the C_j^l and C_j^u found by the code analysis tool comparing them with the simulated number of cycles obtained from the execution of the entire applications on a single ASIP. The values obtained for MJPEG encoder are summarized in columns 2 to 4 of Table 4.6. The values for the SC and ECG case studies are in Appendix A, in Tables A.9 and A.5, respectively. Similarly to the MJPEG encoder case study modeled as a task graph, the *sim* values for most of the tasks are not included in the range $[C_j^l, C_j^u]$ as expected. When, for each task τ_j, we compare the C_j^l with the results obtained from simulation, we have relative errors up to 63% for MJPEG, 97% for ECG and up to 99% for SC for the single tasks. This means that for some tasks we are using upper and lower bound values that are inaccurate. In Section 4.3 we already discussed the possible source of inaccuracy in the code analysis tool. As an additional consideration, we need to mention that the code analysis tool provides better results when we are comparing the execution cycles of the entire application and not the contributions of the single tasks: in this case we have errors up to 16% for MJPEG, 32% for ECG and 18% for SC.

As we did in Section 4.3, we evaluated the relative contribution of each task to the total number of cycles of the application. The results obtained for C_j^l and *sim* values are available in columns 5 and 6 of Tables 4.6 (MJPEG encoder), A.5 (ECG) and A.9 (SC). Once we had the relative contribution of each task to the total number of cycles (for the estimated C^l and simulated *sim*), we calculated the absolute error between them, which is available in column 7 of Tables 4.6 (MJPEG encoder), A.5 (ECG) and A.9 (SC). This evaluation shows which tasks are more time consuming than others. This is reflected also in the simulation results using implemented ASIPs. For our case studies, we got errors up to 3.61% for MJPEG encoder, 14.36% for ECG and 29.06% and for SC. Even with such relevant errors, our DSE with UM is able to identify which tasks are more time consuming than others and determines, also considering the communication between tasks, which task clustering solution should be implemented.

For the MJPEG encoder case study we also verified what happens when we provide as input to our DSE with UM more accurate values for the upper and lower bounds. We took the *sim* values from Table 4.6 and we scaled them of $\pm 30\%$ to get the upper and lower bounds. The *sim* values are used as the mean (μ) of the normal distributions in our UM. Then we ran again our DSE: we found the same task clustering solution (Sol_1) found in Section 6.1. This result is an additional element that suggests that our

UM is not sensitive to relevant errors in the evaluated upper and lower bounds. From our analysis we can infer that the relative size of the tasks is the one influencing the outcome of the DSE. Moreover, this suggests that a designer responsible for defining the upper and lower bounds, can use approximate values and the DSE can still find a promising task clustering solution.

For comparison, we also evaluated the remaining task clustering solutions in Table 6.4 using the new upper and lower bounds. The results obtained are described in Table 6.9. These results confirmed the ones obtained with the less accurate upper and lower bound values. Additionally, we observed that the quantile values ($\delta^{0.5}_{A_{MJPEG}} = P^{-1}(p_{0.5})$, column 6) are very close to the ones obtained through simulation with SH tools (column 8), the maximum relative error between them is 8% (for Sol_1). The quantile values are close to the simulated ones for the way we build the CDF using the sim values as the μ of the normal distributions; however, this indicates that our scheduling analysis using TLA and PA is quite accurate and returns results consistent to the ones obtained with SH simulator.

Table 6.8: Comparison between the number of cycles estimated by the profiling tool [43] and the ones obtained from simulation for MJPEG

Task Name	C_j^l	C_j^u	sim	$\%Tot_{C_j^l}$	$\%Tot_{sim}$	Err $\lvert\%Tot_{C_j^l} - \%Tot_{sim}\rvert$
mainDCT	25,695,360	71,617,920	69,815,040	51.52	55.13	3.61
mainQ	6,401,280	8,868,480	12,150,620	12.83	9.59	3.24
mainVLE	16,212,735	24,024,690	40,942,298	32.51	32.33	0.18
mainVideoOut	1,565,550	1,934,250	3,168,482	3.14	2.50	0.64
Total (cycles)	**49,874,925**	**106,445,340**	**126,635,428**			

6.5 Additional discussion of the results

During the implementation of our DSE with UM we took into account the SH technology used for the experimental evaluation; in fact, having some knowledge about the target technology can produce more accurate results.

For example, we adjusted our algorithm for the evaluation of a clustering solution to be consistent with SH simulator. We added offsets in the schedulability analysis for modeling the time required for starting the execution of the tasks on the ASIPs, for modeling the synchronization time (access to the FIFOs), and also for considering additional bus parameters as the hand-shake time to gain access to the bus and the setup time for transfer the data. These values are stored as offset and can be easily changed or removed according to the technology available.

Moreover, for the inter-processor communication, we considered a message-passing paradigm, in which a processor is responsible for sending data to the next processor. This implies that the processor is busy not only during the execution of the tasks, but also during the transmission of the data. In fact, in our system, we have no DMA for the read and write operations to and from memory[1]. Our tool for DSE accepts as input a flag to specify if the processors are involved in the transfer of data or not.

6.6 Experimental evaluation with a NoC

In this section, we present an example of our DSE with UM that has been extended for evaluating a communication architecture different from a bus.

In particular we consider a NoC architecture that represents the today paradigm for connecting a high number of integrated cores. NoCs offer a structured and reusable communication architecture that bypasses the issues of a bus-based shared architecture. Additionally, their structured architecture allows reducing the design time and the time-to-marked [59]. In the literature there are static schedulability analyses for real-time NoC. There is a large group of approaches based on time-division multiplexing, e.g., the links are shared according to slots of time, to provide schedulability guarantees. Examples of TDM NoCs are dAElite [89], aelite [37], ACROSS approach (for mixed criticality systems) by TTTech [30] and the solutions implemented in [84, 65]. Other approaches implement real-time packet-switching NoC using service disciplines, e.g., using non-blocking routers with rate control in the PEs [103].

We propose a simplified schedulability analysis as a proof of concept to demonstrate the application of the UM to a communication architecture different from a bus and, as future work, we will integrate TDM in our analysis.

Moreover, during our schedulability analysis, we make assumptions about the routing and switching policies. These limitations can be released in a later design phase; once a task clustering solution is established and the micro-architecture configuration for each ASIP is available, it is possible to perform a second DSE to additionally optimize the NoC topology. For example, in a completely customizable system, we can remove unused switches, NIs and PEs and we can further explore different switching and routing policies. Additionally, the output of our optimization can be used as *core communication graph*, i.e., a graph that specifies the PEs and communication demands, indicating the task clusters and the communication latency constraints between each couple of tasks. The *core communication graph* can be used as input for those approaches available in the literature that suppose a pre-defined application mapping and PEs and that focus on the NoC optimization and schedulability analysis [39, 72, 13].

[1] SH technology allows the insertion of a DMA, but it is not available under our University license.

Table 6.9: Comparison of clustering solutions for MJPEG encoder using updated upper and lower bound values

Sol_{ID}	Clusters			$P(\delta_{A_{MJPEG}} < d^{15}_{MJPEG})$	$\delta^{0.5}_{A_{MJPEG}} = P^{-1}(p_{0.5})\ (\mu s)$	*sim* (cycles)	*sim* (μs)	sched
	PE_1	PE_2	PE_3					
1	mainDCT	mainQ, mainVLE, mainVideoOut	-	0.98	437900	79088561	476437.11	yes
2	mainDCT, mainQ	mainVLE, mainVideoOut	-	0.43	505500	85740371	516508.26	no
3	mainDCT, mainQ, mainVLE	mainVideoOut	-	~ 0	744000	124194971	748162.48	no
4	mainDCT	mainQ, mainVLE	mainVideoOut	0.46	502600	83617556	503720.22	no
5	mainDCT, mainQ, mainVLE, mainVideoOut	-	-	~ 0	759700	126635428	762864.02	no
6	mainDCT	mainQ	mainVLE, mainVideoOut	0.36	511700	83636411	503833.80	no

6.6.1 Network model

We assume a NoC with a MESH topology [104] as communication architecture. We select the MESH topology as, to date, is one of the most scalable designs to the increasing number of PEs [105]. The CP model presented in Chapter 2 can be used to model a NoC. An example of CP model for a 2x2 MESH NoC is depicted in Figure 6.7a. We assume that each switch is connected to a PE through a network interface (Figure 6.7b). A switch has four interconnections toward the other switches in the network (north, south, east and west) and an interconnection with the NI (Figure 6.7b). For simplicity, for this schedulability analysis, we assume that each switch allows a single flow of data from an input to an output port[2].

We consider wormhole switching [17] and a static XY routing policy [75]. In wormhole switching, each packet is divided into smaller pieces, called flits (**fl**ow control digi**ts**). The first flit, called header flit, is analyzed by the switch to set up a connection between the incoming port on which the flit arrives and the outgoing port on which the flit is sent; the remaining body flits do not contain any routing information and the switch is responsible for forwarding them according to the header flit, in a pipelined manner. A tail flit is used to terminate the flit forwarding and then a new packet can be processed.

As for the implementation with a bus (Chapters 3 and 5), we assume a message-passing approach, i.e., the exchange of data between tasks assigned to different PEs happens through the exchange of messages and not through the access to a shared memory [105]. We use a static XY routing, because it is deterministic and simple to calculate: all the communication paths between each couple of source and destination PEs can be calculated offline, before starting the DSE. In our model, each flit has a fix size of four bytes and it is the amount of data that can be stored in a switch on each incoming port. We suppose packets with a variable number of flits. Each packet corresponds to a message m_g in the application model. When we process a packet, we add a header and tail flits.

The PEs in the initial NoC platform are ASIPs which micro-architecture configurations have not been defined yet. They are defined after the selection of a task clustering solution, which is established by our schedulability analysis with UM.

The number of PEs is specified by the input CP model. However, we assume a square MESH topology for the initial platform; therefore, we add PEs, if needed. For example, given a CP model with two PEs, we add two extra PEs to obtain a square (2x2) MESH topology.

[2]This is a constraint that will be released in future work allowing multiple transfer if there is no contention of the input and output ports.

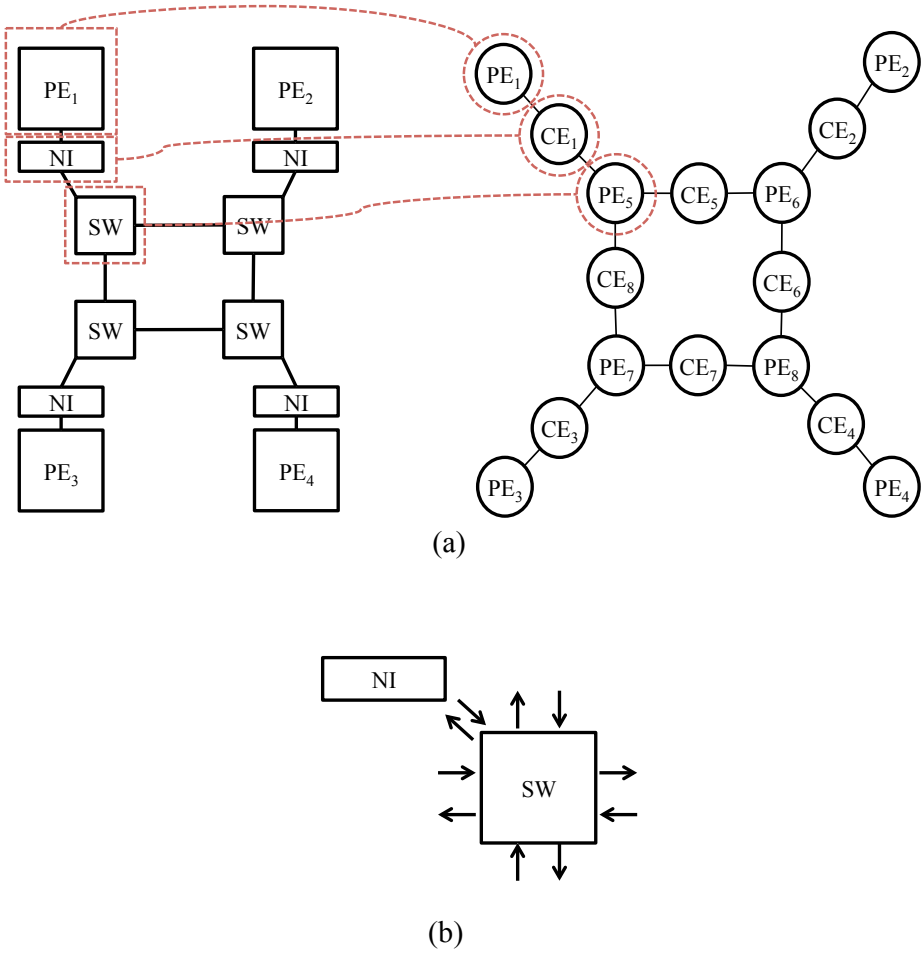


Figure 6.7: Example of CP model for a 2x2 MESH NoC (a) Switch (SW) model (b)

6.6.2 Schedulability analysis

We model the application using a SDFG and we use a static non-preemptive schedulability analysis similar to the one presented in Section 5.2. The application has a deadline, d. The WCET of each task is a stochastic variable and it is modeled by the UM as described in Chapter 2. We know the size (in bits) of each message. We suppose that a message corresponds to a packet with a variable number of flits. Given the flit size, we calculate the number of flits contained in each packet for a specific message. The schedulability analysis uses Monte Carlo simulation to enable the analysis with the UM of the tasks.

Using the XY routing policy, we estimate the communication path between each couple of PEs before starting the DSE. Each communication path include the source PE (and the attached NI), the set of traversed switches and the target PE (with its NI). Knowing the path between each couple of PEs, we can determine the transmission time C_{m_g} associated to each message m_g. As for the bus, the C_{m_g} of each message is *not* a stochastic variable.

In order to schedule the SDFG of the application to the CP model of the NoC, we modify the message representation in the model of the application. An example is depicted in Figure 6.8. First we calculate the number of flits in each message and we add an header and tail flits. m_1, in Figure 6.8a, contains three flits (a header, a payload and a tail flit). m_1 is transferred from PE_1 to PE_2 using two switches (for simplicity we add the NI traverse time to the WCET of the task). Therefore, m_1 is split into two sub-messages to model the transfer through the network: $m_1^{PE_5}$ and $m_1^{PE_6}$. Additionally we need to split each one of these messages into the corresponding number of flits as specified in Figure 6.8b. This representation allows considering the pipelined transmission of the flits through the network.

The modified version of the SDFG is dynamically calculated for each evaluated task clustering solution as it depends on the task clusters, the number of flits per message and the number of traversed switches. It is used as application model for the schedulability analysis that is performed as described in Chapter 3. We take into account the hardware resource contentions and the flit propagation happens only when a switch is not involved in another transfer[2]. The contention is resolved based on the message and task priorities (statically assigned). The output of the schedulability analysis is the probability of a task clustering solution to meet the application deadline. The main advantage of the NoC is that is can reduce the communication time exploiting multiple communication paths and introducing pipeline parallelism also in the communication architecture. As shown in Figure 6.8b, a PE can transmit multiple flits through the network without waiting for them to be received. In fact the buffers in the NIs and in the switches can store the flits and allow a pipelined transmission.

6.6.3 Results

We tested the tool with the MJPEG encoder application already used in Section 6.1; the SDFG model is depicted in Figure 6.2; we used the upper and lower bounds for the WCET of each task and the size for messages listed in Tables 6.2 and 6.3. We also used the same input constraints: a frequency $f = 166MHz$ for the PEs and the network and a deadline $d = 500,000\mu s$ for the elaboration of 15 frames. The initial platform is the 2x2 MESH NoC represented in Figure 6.7.

As the design space was limited (i.e., 256 task clustering solutions, without considering

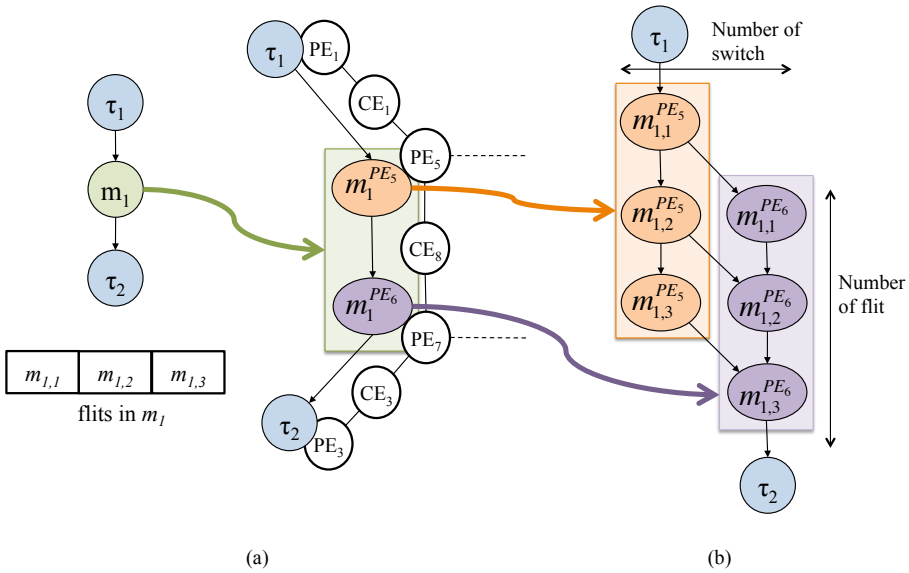


Figure 6.8: Clustering of a message, m_1, on the NoC

the source and sink tasks which WCET is set to zero), we could run an exhaustive DSE. The cumulative distribution functions obtained from the DSE are shown in Figure 6.9. Between the solutions found, we chose the one with the highest probability of meeting the deadline, the smallest quantile value at $\delta^{0.5}_{A_{MJPEG}} = P^{-1}(p_{0.5})$ and the smallest number of PEs and switches. The selected solution is summarized in Table 6.10. The solution found is the same one already found using a simple bus; this means that the communication network is not the bottleneck for the MJPEG case study, which WCET depends on task *mainDCT* that is the slowest one. Furthermore, we observed that the quantile value $\delta^{0.5}_{A_{MJPEG}} = 293,302\mu s$ is comparable (0.2% difference) to the one estimated with the bus ($292,700\mu s$) that also confirmed that the NoC and the bus have similar performances for the MJPEG encoder application.

The MJPEG encoder, due to its limited size cannot exploit the potential of a NoC as the performance of the platform depends on the processing time that is much higher than the time spent in the exchange of messages; however, our goal with this example was to show that the schedulability analysis with the UM could be extended to work with communication architectures different from a bus and, in particular, that it can be applied to a NoC. As future work, more applications and larger NoCs should be considered to verify the performance improvements obtained from the use of a NoC as communication network.

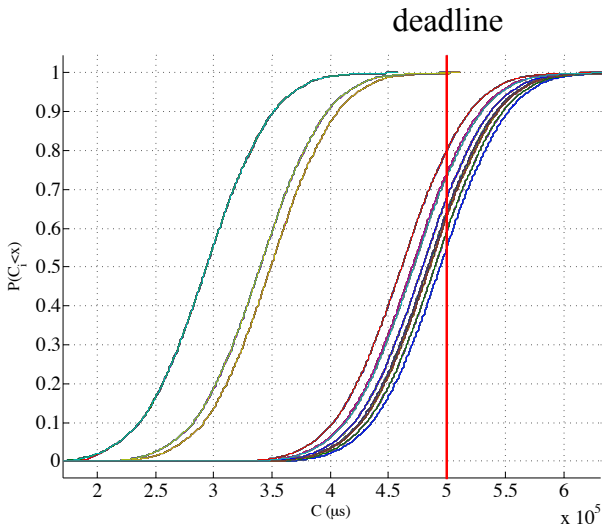


Figure 6.9: Output CDFs for MJPEG application clustered on a MESH NoC

Table 6.10: Clustering solutions for MJPEG encoder with MESH NoC

Clusters		$P(\delta_{A_{MJPEG}} < d)$	$\delta^{0.5}_{A_{MJPEG}} = P^{-1}(p_{0.5})$ (μs)
PE_1	PE_2		
mainDCT	mainQ, mainVLE, mainVideoOut	~ 1	293302

6.7 Summary

In this chapter we described the MJPEG encoder, the ECG and SC case studies modeled as a SDFG. We ran our DSE with UM, with the schedulability analysis (TLA and PA) described in 5.2. We demonstrated that with our analysis, we could select the proper task clustering solution for implementation.

We verified that the SDFG model allows a smaller level of granularity and, hence, better performance can be reached compared to the task graph model.

We evaluated the influence of the upper and lower bounds in our DSE and, as we did for the case studies modeled as task graph, we concluded that our analysis is not sensitive

to big variations in the C_j^l and C_j^u. However, we verified that with more precise values for the upper and lower bounds, we could obtain estimated values (the quantile function at 0.5) very close to the ones obtained with SH technology (less than the 8% of error for the MJPEG case study).

In this chapter, we also presented a small example to show that our schedulability analysis can be applied to more complex communication architecture than a bus.

CHAPTER 7

ASAM project

In this chapter, we present the integration of our DSE with UM to ASAM [7], a semi-automatic design flow for the implementation of multi-ASIP platforms. ASAM is a project in the framework of the European ARTEMIS Research Program and ARTEMIS Joint Undertaking that completed in February 2014. Its goal is to provide tools and a design flow for the design and synthesis of ASIPs and multi-ASIP systems. ASAM design flow takes into account the constraints of MPSoC design including power, performance and area. It is out of the scope of this thesis to describe the details of the ASAM project and we consider only a subset of the design flow. Specifically, we focus on the two tools that we created for the macro-architecture DSE and on their integration within the ASAM flow. The first tool is used in the very early phases of the design when there is no information about the platform composition (this includes our DSE with UM). The second one is used in a later phase of the design flow and includes a multi-objective DSE to select within a group of available micro-architectures, the proper micro-architecture for each task cluster.

This chapter is organized as follows. In Section 7.1, we give an overview of ASAM design flow, then in Section 7.2 we introduce Compaan tool for the application partitioning, the micro-architecture DSE tools developed at the Technical University of Eindhoven and the SH technology for the design of ASIPs. In the same section, we describe the tools that we implemented for the macro-architecture DSE. Finally in Section 7.3 we apply the described part of the ASAM flow to the ECG case study [93].

7.1 ASAM design flow

The ASAM project has been carried out with the collaboration of multiple partners from industry and academia. In this section, we provide a high level description of the project and of the parts of the flow that are relevant for the macro-architecture DSE that we implemented (Figure 7.1). The ASAM design flow takes as input the C code of an application A_i, the design constraints (e.g. application deadline, maximum area) and the CP model of the initial platform, i.e. a bus based platform with PC_{max} PEs. It produces as output a multi-ASIP platform optimized for A_i.

All tools communicate exchanging files in a XML format, except for the interfaces towards SH tools, in which the files needed by SH are generated (TIM and HSD formats as presented in Section 7.2.4). Examples of the XML files generated, with a short description for each of them, are available in Appendix B. Figure 7.1 depicts the exchange of data between the tools.

Compaan extracts a KPN model from the application C code. It also produces an XML file with a list of the tasks and edges between them (application model).

The *code analysis tool* (Phase 1 of micro-architecture DSE) elaborates the application C code and returns an estimation of the upper and lower bound values of each task. These values are produced considering a profiled execution of the application. The XML file with the application model (both task graph and SDFG model are accepted) is annotated with the upper and lower bounds and the number of firing of each task.

Our first tool for macro-architecture DSE is then invoked. If the SDFG model is used, the macro-architecture DSE automatically calculates the consumption and production rates for each task. We named this tool *probabilistic DSE* to denote the use of the UM. The probabilistic DSE explores the space of task clustering solutions using the schedulability analysis with UM and returns the clustering solution with the highest chance to meet the deadline.

Then, for each of the task cluster found, the *micro-architecture definition tool* (Phase 2 of micro-architecture DSE) is invoked. It uses an ArrayOL model [32] of each task (obtained from the polyhedral model generated by Compaan) and determines a number of possible micro-architectures for each task cluster. For each micro-architecture, it returns an estimation of area (in μm^2) and performance (in cycles) for task.

Finally, we can run our second tool for macro-architecture DSE, named *deterministic DSE*, which uses the output produced by Phase 2 to perform a schedulability analysis of the application running on the multi-ASIP platform and to calculate the total area. We evaluate the different micro-architectures proposed by Phase 2 and select the ones that allow meeting the input constraints.

ASAM design flow also includes the optimization of the energy and a specific optimization of the communication architecture that we omit as it is out of the scope of this thesis. The last step of the design flow is the implementation of the multi-ASIP system using SH technology. This step is not automated and it requires the manual intervention of the user to map the C code of the application to the ASIPs (e.g. insertion of APIs for synchronization and exchange of data between processors).

7.2 Tools in ASAM design flow

In this section, we describe the tools developed for ASAM design flow and that we used in Sections 4.2 and 6.1 for our case studies. Compaan compiler and SH toolchain are commercial tools, while the micro-architecture DSE (Phase 1 and 2) were developed at the Technical University of Eindhoven (TU/e). At the Technical University of Denmark, we developed the two tools for macro-architecture DSE.

7.2.1 Compaan Compiler

Compaan Compiler (part of Compaan Design) [1] is a commercial tool for C-to-dataflow conversion. It takes as input the sequential C code of an application and returns its KPN applying polyhedral models that are based on an algebraic representation of the code [90]. The KPN model allows expressing the C code in terms of tasks, data and pipeline parallelism. It also computes the firing rates for each task.

In ASAM design flow, we use Compaan compiler for the partitioning of the application into tasks. For the analysis with Compaan, the generic C code of an application needs to be modified as follows [50]:

- selection of a *top-level function* (indicated by a *pragma compaan_procedure*) that is the one analyzed by Compaan compiler to generate the KPN;
- identification of the *kernel* functions invoked inside the *top-level function* that will become the nodes (i.e. tasks) of the KPN;
- modification the *top-level function* so that it contains affine nested loops.

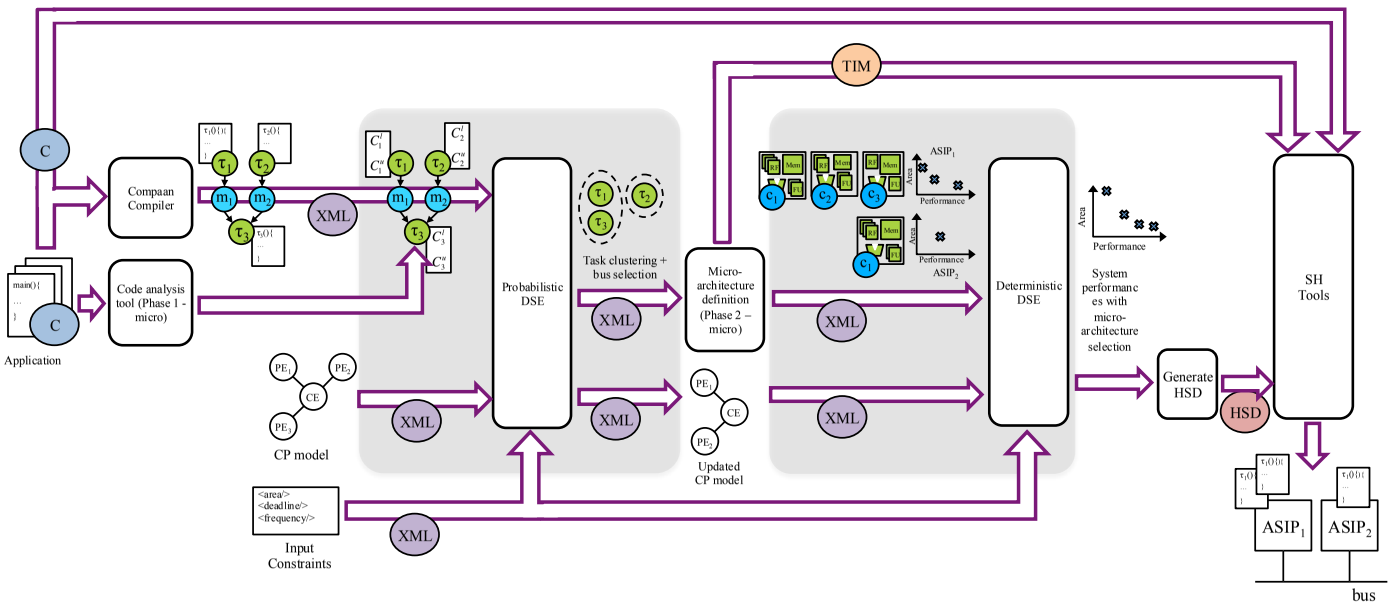


Figure 7.1: ASAM design flow and interfaces with macro-architecture DSE

7.2.2 Code Analysis tool - Phase 1 of micro-architecture DSE

The code analysis tool [43] allows estimating the number of cycles of the overall application performance and of each function. The tool uses LLVM compiler framework [64] for translating the input application from C to a lower-level intermediate representation (IR). The IR is used to estimate the upper and lower bounds, C^u and C^l for each function in the C code. This estimation is done using real profiled information extracted from the execution of the C code. The estimated number of cycles, for each task identified by Compaan, is calculated summing the number of cycles of each function invoked inside the task code as specified by the call graph of the application. C^u is the expected number of execution cycles when the task is executed on a scalar processor. C^l is the expected number of cycles when the task is executed on a virtual processor according to an as soon as possible (ASAP) scheduling without architectural constraints. In Sections 4.2 and 6.1, for our case studies, we observed a discrepancy between the estimated upper and lower bounds and the number of cycles obtained when compiling the same code on SH ASIP. This discrepancy can derive from complex operations (e.g. load and increment pointer) that are automatically detected and used by the SH compiler; additionally LLVM and SH compiler perform different kinds of optimization on the target application (e.g. different loop optimization performed).

As mentioned in Section 4.2.1, the code analysis tool does not return a theoretical estimation of the WCET; it returns an estimated number of cycles of a profiled execution of the application. Our probabilistic and deterministic DSE would both expect the WCET of each task, in fact, with static scheduling, to provide schedulability guarantees we should use the WCET. However, we tested the code analysis tool with different set of input data for our case studies and we considered the ones producing the longest execution time. We know that this is not a theoretical WCET, but we consider this value good enough for our DSE. The same consideration applies to the Phase 2 of the micro-architecture DSE that is described in the following section. In fact the performance values returned by Phase 2 are calculated with a similar approach.

7.2.3 Micro-architecture DSE tool - Phase 2 of micro-architecture DSE

The micro-architecture DSE [43] performs a data-oriented software and hardware co-design to decide the ASIP micro-architecture. It takes as input a task cluster and the polyhedral model generated by Compaan of each task in the cluster. The micro-architecture DSE uses a data-oriented polyhedral representation of the application [32], which is used to rapidly evaluate the task's code and the corresponding hardware architectures. It performs a DSE in order to select the loop transformations that are used to infer the ASIP micro-architecture. This evaluation happens through a static analysis of

the data-oriented representation. The micro-architecture DSE decides the data memory size and the data parallelism through vectorization or usage of multiple issue slots (IS). Each micro-architecture is composed of a selection of ISs taken from a library and is characterized by an estimated area (in μm^2) and performance per task (in cycles).

Examples of loop transformations are loop unrolling or loop fusion (merging two or more loops in a same iteration space, reducing the loop control [48]). Depending on the loop transformation applied, the micro-architecture DSE establishes if the ASIP micro-architecture should contain ISs with vector instructions and how many ISs are necessary to exploit the data parallelism.

For each task and micro-architecture, it adds information about the estimated area and the performance of the tasks. For each task, it specifies the start and end cycles (with respect to an initial offset=0). As Phase 2 performs DSE considering one cluster at a time, it has no visibility of the other clusters and cannot determine if synchronization with tasks associated with different ASIPs is needed. Moreover, in case of micro-architecture optimization as loop fusion, multiple tasks in a cluster are executed in parallel. Therefore, the next phase in the ASAM flow, i.e. the deterministic DSE, to perform a schedulability analysis of the entire system, needs to know the start and end cycles for each task (a latency value is not sufficient). More details are available in Section 7.2.5.2.

7.2.4 Silicon Hive technology

SH tools [63] allow the definition of a multi-ASIP platform in which the ASIPs are connected to each other and to external memories through a hierarchy of buses.

SH provides TIM language for the description of the ASIPs and Hive System Description (HSD) language for the definition of the platform. Both TIM and HSD are proprietary hardware description languages and have their own compiler. TIM Compiler checks the design rules for the ASIP and generates a pre-processed version of the ASIP design description, while HSD Compiler generates a pre-processed version of the multi-ASIP design description (including buses and external memories).

Additionally, SH provides an ANSI-C compiler, HiveCC, that can be used to compile the application code and to generate executable code for the multi-ASIP platform described by TIM and HSD languages. HiveCC includes also a simulator that returns a cycle-accurate estimation of the application running on the multi-ASIP platform.

The SH ASIPs are single-threaded processors and have a VLIW architecture that is configurable depending on the functionalities required by the tasks. They can contain scalar or vector functional units and one or more ISs. Each IS contains a set of

functional units available in a library that are connected to local or external memories, register file(s) and input/output FIFO ports. Figure 7.2 shows an example of the micro-architecture of a SH ASIP [62].

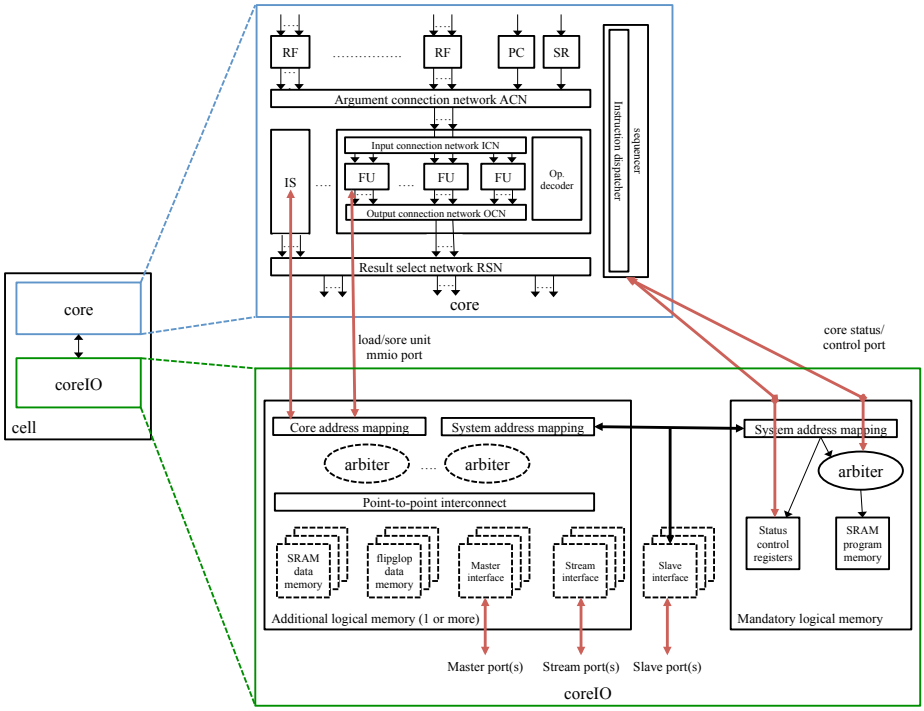


Figure 7.2: Example of micro-architecture of a SH ASIP [62]

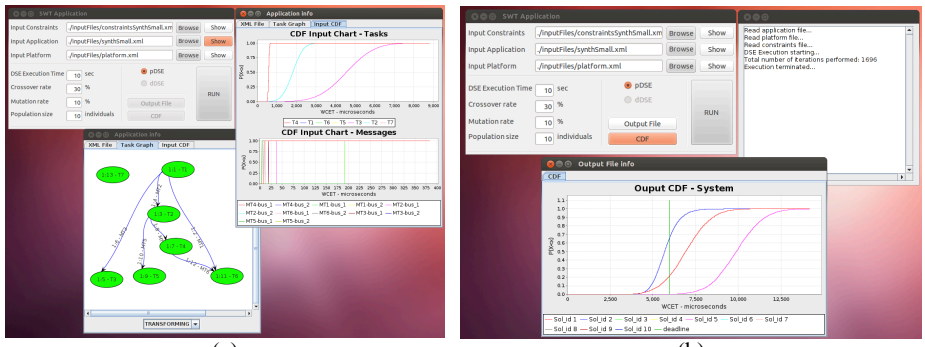
For the execution of the C code on SH processors, it is necessary to edit the code as follows. First, the code of the different tasks in a task cluster is merged in a single function that corresponds to the task cluster's code. Then, we add the APIs for the access to the local or external memories and for the transfer of data. The multi-ASIP platform, to be simulated, requires also an *host* processor that is in charge of loading the code of the task clusters on the ASIPs, starting their execution and also dispatching the input data or verifying the output, if necessary.

In ASAM and in the case studies described in this thesis, we impose some design constraints that consider only a subset of SH functionalities. We add two FIFOs between each couple of ASIPs that needs to exchange data. The FIFOs are used only for synchronization purposes while the data are transferred on the bus. We use a single 32-bit bus for interconnecting the ASIPs. For the synchronization of the ASIPs with the host processor, we need to provide the host with the access to the FIFO ports of the ASIPs. The host does not have any FIFO ports; it has only access to the system bus

(and to the hardware components connected to the bus). Therefore, to guarantee the synchronization host-ASIP, we use a hardware block, called FIFO adapter, that works as an interface between the ASIP FIFO ports and the bus. Using the bus and the FIFO adapter, the host can synchronize with the ASIPs.

7.2.5 Macro-architecture DSE

In this section, we describe the probabilistic and deterministic DSE tools developed for the macro-architecture DSE. We specify how they interface the other tools of the ASAM design flow. We implemented two versions of the tools, one that works through command line (used inside the ASAM flow) that support both the task graph and SDFG application models and one with a GUI (to use it as a stand alone tool) that support only the task graph application model. Both versions have been implemented in Java. Screenshots of the GUI of the probabilistic and deterministic DSE tools are shown in Figures 7.3 and 7.4.

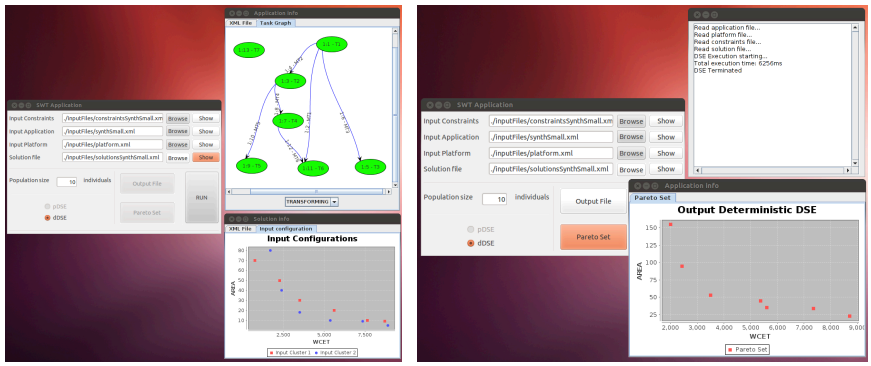


(a) (b)

Figure 7.3: GUI of the probabilistic DSE tool, input (a) and output (b)

7.2.5.1 Probabilistic DSE

The probabilistic DSE is executed in the early phases of the design flow when there is no information about the platform and its composition. The inputs available are the graphs (SDFG or task graph) of one or more applications, their design constraints and the CP model of the initial platform (in XML format). In Listings B.1, B.2 and B.3 there are examples of input files. The purpose of this DSE is to identify which tasks should be grouped into the same ASIP without a precise knowledge of the underlying hardware (ASIPs and their communication architecture). The probabilistic DSE implements the DSE with UM described in Chapters 3 and 5. The probabilistic DSE tool accepts as



(a) (b)

Figure 7.4: GUI of the deterministic DSE tool, input (a) and output (b)

input also the SSEA parameters: population size, mutation and crossover probabilities and the time the exploration should run.

The output is a set of task clustering solutions that have high chances to meet the deadline when continuing through the successive phases of the ASAM flow. The task clustering solution with higher chances to meet the deadline is sent to the micro-architecture DSE (Phase 2) for ASIP optimization (an example is available in Listing B.4).

The micro-architecture DSE (Phase 2) requires also a KPN model for each task cluster. We obtain the KPN model of a cluster extracting it from the KPN model of the entire application (generated by Compaan). For each cluster, we take the KPN nodes that correspond to the tasks in the cluster and all edges connected to them. In most of the cases, extracting nodes and edges from a KPN model is not enough to get a new KPN. In fact, as we split the tasks among different clusters, some input/output messages of the tasks are unconnected. Therefore, we add dummy tasks that have no computation associated to complete the KPN.

7.2.5.2 Deterministic DSE

The deterministic DSE is the second tool that we developed for macro-architecture DSE. It is invoked in a later phase of ASAM design flow, after the micro-architecture DSE. At that point in time, we have more precise information about the possible micro-architecture implementation for each ASIP: this allows substituting the UM of each task, with a set of more precise performance values. The performance is not expressed as a WCET value, but as a start and end times. The deterministic DSE uses these values to run the schedulability analysis. Each performance value assigned to a task,

corresponds to a micro-architecture configuration. The micro-architecture DSE returns also an estimation of the area for each micro-architecture. Therefore, we can also estimate the area for the entire system.

The deterministic DSE runs a multi-objective DSE; we use the NSGA II algorithm [19] implemented in the Java library jMetal [4]. The objectives to optimize are the area and the scheduling length. The algorithm terminates after a number of iterations that is specified as input. It accepts as input also the population size, the crossover and the mutation probabilities (only in the command line version of the tool).

The deterministic DSE can perform a schedulability analysis using both a task graph and a SDFG application models. The schedulability analysis follows the same steps presented in Sections 3.3.1 and 5.2 for the scheduling with UM. The main differences are that instead of Monte Carlo arrays with n samples, we use single values and that the deterministic DSE has to consider the optimization performed by the micro-architecture DSE. This last point makes the schedulability analysis more complex. When the micro-architecture DSE performs loop fusion, it means that the tasks are merged and executed in parallel as a single thread. However they do not start at the same time: they are shifted of a certain number of cycles. The same applies to their completion time. Moreover, it is possible that the merged tasks have different data dependencies and, as they have to execute as a single task, they have to wait for the dependencies of all merged tasks to be satisfied. This is also the reason for having the start and end cycles for each task and not a WCET value, as we need to know which tasks have to be scheduled in parallel as a single task.

The output of the deterministic DSE is a file containing the id of the selected micro-architecture configurations of each ASIP with the estimation of area and performances for the entire system. As the deterministic DSE uses a multi-objective DSE, it returns not a single solution but a Pareto set of solutions. Therefore, we ask the designer to indicate which design parameter he wants to optimize (e.g. area or performance). We use this information for selecting a single solution when there are multiple solutions meeting the constraints. Listing B.6 contains an example of output of the deterministic DSE.

The design flow completes when a platform description that meets the input constraints is available, then it is possible to synthesize the entire multi-ASIP platform using SH tools.

7.3 Experimental evaluation

In this section, we present the results obtained for the ECG case study elaborated with the portion of the ASAM design flow previously described. We consider the SDFG model for the ECG application. Some of these results are also presented in Appendix A.2.1 where there is an extended analysis of the output of the system-level DSE with UM for the ECG.

The input constraints for the ECG are summarized in Table 7.1. We suppose the elaboration of 10,000 samples and a deadline of $16s$. The XML file with the input constraints that is used as input to the ASAM design flow is shown in Listing B.1. The initial platform model (CP model) is in Listing B.2. We consider a bus-based platform with a maximum of 2 PEs, $PC_{max} = 2$. Moreover, we assume that the parameter to minimize is the area (used by the deterministic DSE to select a design point from a Pareto set of solutions).

The first step of ASAM design flow is the elaboration of the C code with Compaan compiler. The KPN model produced by Compaan is depicted in Figure 7.5a. Starting from this model, we derived the SDFG model in Figure 7.5b.

The second step is the elaboration of the input C code with the code analysis tool (Phase 1 of micro-architecture DSE) that produces the upper and lower bounds of the WCET. The XML files with the application model annotated with the upper and lower bounds value is shown in Listing B.3. The upper and lower bounds, C_j^u and C_j^l, for a single iteration of each task τ_j, are summarized in Table 7.2.

Table 7.1: Input constraints for ECG

$d\ (\mu s)$	$f\ (MHz)$	max. area (μm^2)	PC_{max}	*Bus type*
16,000,000	1	8,000,000	2	b_{32}^1

Table 7.2: C values for ECG (average number of cycles for a single iteration of the task)

C	$lowpass$	$highpass$	$derivative$	$square$	$integrative$	$detect$
C_j^u	46	85	1	1	2038	16
C_j^l	41	46	1	1	1084	10

As a third step, we run our probabilistic DSE that performs a schedulability analysis and returns the task clustering with higher chances to produce a schedulable solution

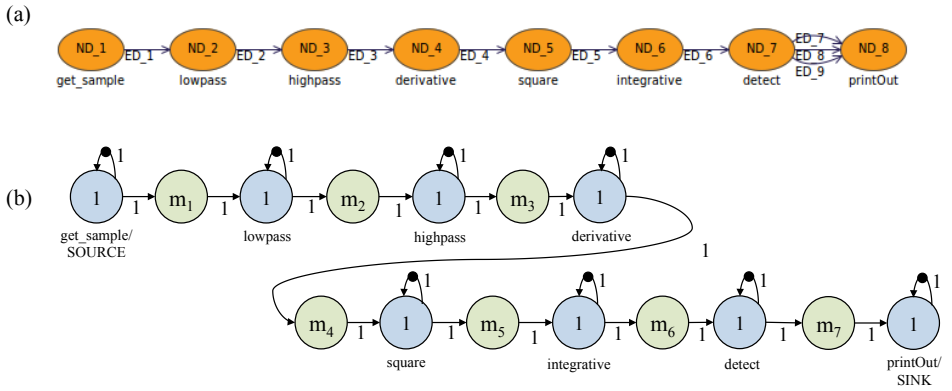


Figure 7.5: KPN model generated by Compaan for the ECG (a) and corresponding SDFG model (b)

once the final platform is available. The clustering solution found is depicted in Figure 7.6a and the corresponding CDF modeling its performance is in Figure 7.6b. The file with the clustering solution produced as output of the probabilistic DSE is available in Listing B.4.

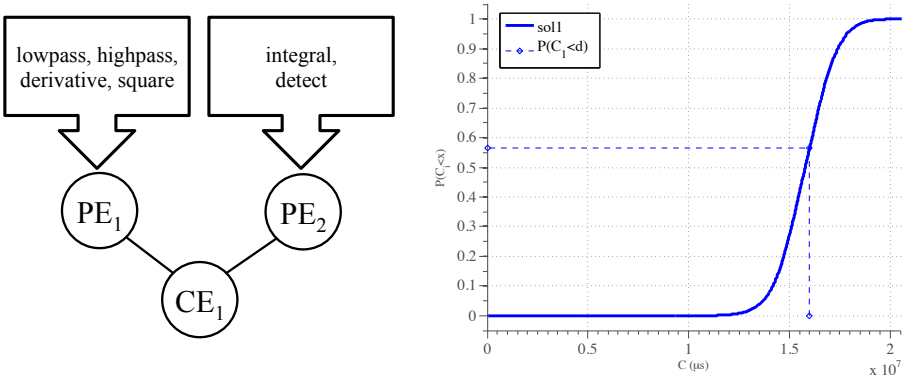


Figure 7.6: Task clustering solution (a) and its CDF (b) produced by the probabilistic DSE

The fourth step is the execution of the micro-architecture DSE (Phase 2). It considers one cluster of tasks at a time and it defines a number of possible micro-architectures for each ASIP. The results obtained by Phase 2 for clusters PE_1 and PE_2 are depicted

in Figure 7.7; c_h indicates the micro-architecture configuration h. Phase 2 finds four possible micro-architecture configurations for PE_1: three of them have scalar micro-architectures with three ISs (the difference in the area derives in the number of register files included) and one (c_2) has a vector micro-architecture with 2 scalar ISs and one with vector instructions. One scalar micro-architecture configuration is found for PE_2. The details about the micro-architecture configurations are not available as the library of ISs used is proprietary and not public available.

For each micro-architecture configuration the corresponding TIM description is available, so that it can be processed by TIM compiler (presented in Section 7.2.4). The performance values in Figure 7.7 consider the execution of all iterations of the tasks and are only indicative as they are calculated building a schedule for a single cluster (without considering the data dependencies with the tasks on other clusters). The output produced by the micro-architecture DSE for ECG case study is available in Listing B.5. The WCETs estimated by the micro-architecture DSE, for each task, are extracted from the range of values delimited by the upper and lower bounds calculated by the code analysis tool. As the estimations provided by the code analysis tool might be subject to inaccuracies (as presented in Chapters 4 and 6), the results produced by the micro-architecture DSE suffers from these errors and extra simulations might be required to obtain more accurate WCET values. The selected WCETs depend on the theoretical instruction level parallelism (ILP) that is provided by each micro-architecture configurations.

The fifth and final step of the ASAM design flow that we consider is the deterministic DSE. It performs a multi-objective DSE using the information provided by Phase 2. It returns a set of Pareto points (Figure 7.8) that meet the design constraints. As the parameter to minimize is the area (specified as input), the deterministic DSE tool selects solution $solDet_2$ with $area = 7,767,160\mu m^2$ and $WCET = 12,550,167\mu s$.

This part of the flow executes in less then twenty minutes. The flow for the ECG case study is completed building the TIM description of the selected ASIPs (generated by Phase 2), writing the HSD description of the multi-ASIP platform and mapping the application code to the processors. Then we can run the simulation using HiveCC (as presented in Section 7.2.4). The simulation returns $13,790,796$ cycles or $13,790,796\mu s$ at $f = 1MHz$. This implies an error of 9% between the performance estimated by ASAM flow and the one obtained with SH tool. We cannot calculate the error for the area, as we do not have the real area measurement. However, we assume that the area values estimated by Phase 2 are built considering a precise definition of the micro-architecture configurations and are good enough to perform a qualitative comparison between different micro-architectures.

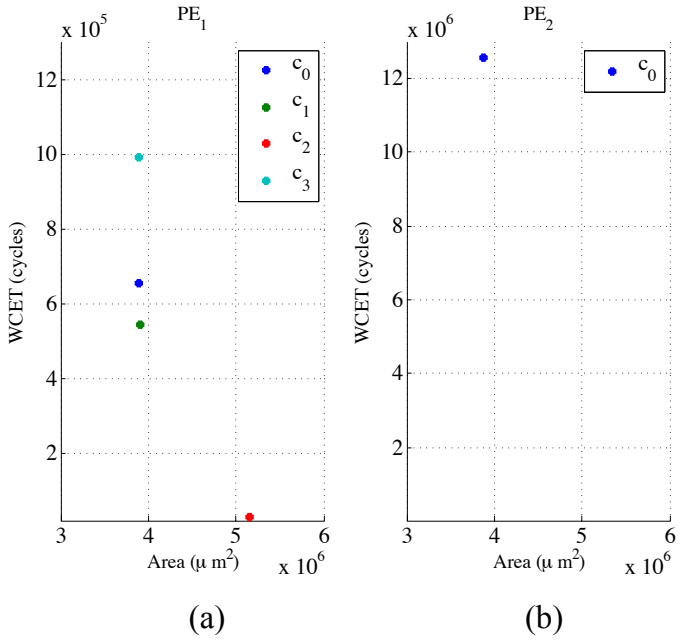


Figure 7.7: Output of the micro-architecture DSE for clusters PE_1 (a) and PE_2 (b)

7.4 Summary

In this section we presented part of the ASAM design flow. ASAM offers a semi-automatic design flow for the implementation of multi-ASIP platforms customized for the execution of specific applications. In particular we presented how the tools for macro-architecture DSE can be integrate in a design flow to support the automatic generation of multi-ASIP platforms. We developed two tools, the probabilistic and deterministic DSEs. The first one is used in the early phase of the design flow and it implements the DSE with UM presented in Sections 3.3.1 and 5.2. The second one is used in a later phase of the design and evaluates the platform performance and area once a set of micro-architecture configurations has already been defined for each ASIP. We also showed the results for the ECG application produced by the presented ASAM tools.

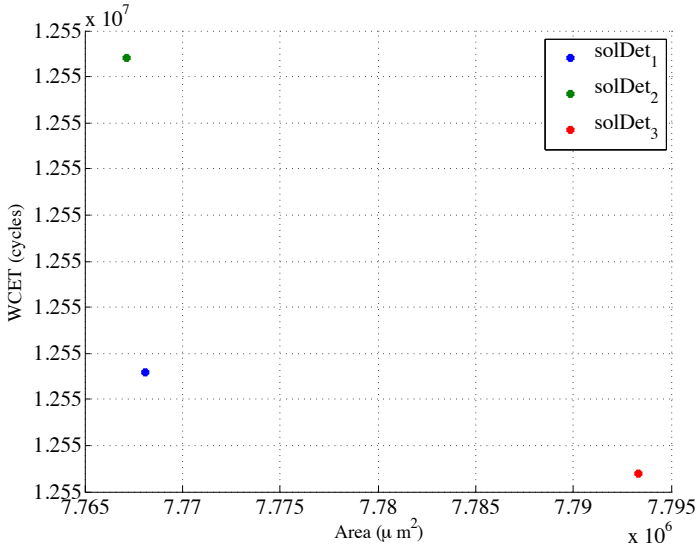


Figure 7.8: Output of the deterministic DSE

CHAPTER 8

Uncertainty model and task similarities

This section presents an extension of the application of the UM for the design of multi-ASIP platform. As described in Chapter 1, an ASIP micro-architecture is configured to execute a cluster of tasks that corresponds to a set of functionalities: the micro-architectural parameters such as number and size of registers and memory blocks and number of functional units are tuned for the task cluster. In the previous chapters we considered the influence of these parameters on the WCET of the tasks assigned to an ASIP. In this chapter we also analyze how a task clustering influences the area of the datapath (i.e., the portion that performs computations) of the ASIP.

We describe an approach to derive a platform defined by a task clustering solution such that there is a simultaneous minimization of the *area cost* and a maximization of the schedulability probability of the applications. The *area cost* is different from the *platform cost* defined in Chapter 3, where we use it to indicate the number of ASIPs in the platform. In this chapter, the minimization of the *area cost* consists in the optimization of the ASIP datapath area considering the task similarities: two tasks are said to have similarities if some of the operations constituting the tasks are identical and, hence, they can share the same datapath resources. We exploit task similarities to design cost-efficient platform solutions by using resource sharing techniques employed in the synthesis of Custom Instruction Set Extensions (ISEs) [106, 107], where the design of the datapath is optimized considering the hardware resources that can be shared by two tasks with similar operations.

Additionally, we consider event-triggered (ET) [51] real-time applications. In the ET approach, all activities are initiated whenever a significant event occurs. ET systems require a dynamic scheduling strategy where an appropriate task is initiated in response to an event; this also means that we are considering ASIPs that can implement a dynamic preemptive scheduler. We use fixed-priority preemptive scheduling (fpps) to schedule tasks in the applications.

The contents described in this chapter have been published in [28]; the chapter is organized as follows. In Section 8.1, we present the system model; as we use a different scheduling policy and different assumptions compared to Chapters 2 and 5, we redefine the platform and application models for clarity to the reader. Then, we define the problem and explain how task similarities are used for the area optimization (Section 8.2). In Section 8.3, we discuss the multi-objective metaheuristic that we apply. We present the experimental results for a set of benchmark applications in Section 8.4.

The tool described in this section has been implemented in MATLAB 2011 and using its Global optimization toolbox.

8.1 System model

The system consists of multiple applications and the underlying hardware platform. The platform can be modeled using the CP model and consists of PEs interconnected through a bus. A PE is denoted by PE_k. The PEs correspond to ASIPs that we implement accordingly to the tasks that they run. Although we consider only ASIPs in the rest of the chapter, our method can handle the inclusion of other processors, as GPPs and DSPs.

Each application is modeled as a task graph $A = (V, E)$, where V is a set of nodes that represent the tasks and E is the set of edges that represent the communication between the tasks. Each edge corresponds to a *message*, i.e. the data exchanged between a couple of tasks. Each message has associated the number of bits of data to transmit (its transmission time is defined by the bus type). A task i is denoted by τ_i. Each task τ_i has a period T_i. When there are multiple input applications, we aggregate the tasks in all the applications into one global task set. For all the applications, we want to meet a global deadline (which is explained later in Section 8.2.3). The area required for the implementation of each task τ_i on an ASIP is denoted by a_{G_i}, where G_i is the data flow graph (DFG) model associated to the task (as presented in Section 8.2.1).

As described in Chapter 2, the WCET C_i of a task τ_i is modeled as a stochastic variable, which probability distribution is described by the UM. C_i is a normally distributed stochastic variable that is defined using an interval of WCETs delimited by the upper

and lower bound values, C_i^u and C_i^l. Therefore, we define the task information tuple for every task τ_i as $\Gamma_i = \{C_i^u, C_i^l, a_{G_i}, T_i\}$.

8.2 Problem formulation

Given multiple applications where each application is modeled as a task graph, the task information tuple Γ_i for each task τ_i in each application, a global deadline d and a set of PEs, the problem is to design a multi-ASIP platform that simultaneously minimizes the area cost and maximizes the schedulability probability of the applications.

We perform a multi-objective DSE to identify the task clustering solution that has high chances of meeting the deadline and minimizes the *area cost* (i.e. platform area). We use the UM together with a fpps analysis to determine the *schedulability probability*, i.e., the probability of a task clustering solution to meet the deadline (Section 8.2.3). As the WCETs of the tasks are modeled as stochastic variables, each task clustering solution is evaluated using a Monte Carlo simulation (MCS) loop. Each MCS loop iteration performs a schedulability analysis to verify if a solution is schedulable (Section 8.2.3). The output of all iteration of MCS are combined to generate the schedulability probability of a task clustering solution. For the area minimization, we use the *area estimation model* presented in Section 8.2.1 that is based on *task similarities*. The output of the multi-objective DSE consists of the number of ASIPs required and the cluster of tasks on each ASIP.

8.2.1 Area Estimation Model

The area estimation model used to reduce the area cost is based on task similarities that influence the ASIP datapath area and the WCET uncertainty model.

The source code associated to a task τ_i can be described by a DFG. An example is available in Figure 8.1a. A DFG consists of nodes that perform some operations such as addition, multiplication, bit shifting etc. There are many approaches in the literature that, starting from a DFG representation of the C code, derive the corresponding datapath [80, 34, 53]. In particular we look into techniques based on datapath merging: the DFGs of multiple tasks can be merged together and then a single data path for all tasks can be implemented. The datapath merging enables the reuse of hardware blocks (i.e., functional units and registers) and interconnections (at micro-architectural level) by identifying similarities among the DFGs, and produces a single datapath that can work for each DFG. Using datapath merging it is possible to minimize the amount of hardware blocks and interconnections in the datapath [88].

Let us consider two tasks τ_1 and τ_2 represented by their DFGs G_1 and G_2 as shown in Figure 8.1a. The clustering of two tasks is represented as the merging of the DFGs G_1 and G_2 and the merged graph (shown in Figure 8.1b) represents the datapath of the ASIP for the task cluster comprising of τ_1 and τ_2. The shaded parts in Figure 8.1a depict the task similarities between tasks τ_1 and τ_2. The task similarities derive from the identical set of nodes (that perform the same functionalities) and edges connecting these nodes in the DFG of the tasks. In Figure 8.1a, the nodes $\{o_1,o_2,o_4,o_5\}$ in G_1 are similar to nodes $\{O_1,O_2,O_5,O_7\}$ in G_2. When we cluster together two tasks, similar nodes are merged together (as shown in Figure 8.1b) to reduce the area of the final ASIP datapath.

The existing graph merging techniques for datapaths can be optimized for area [71] and latency [106]. In [71], compatibility graphs are used to detect the similar nodes in the merged DFGs and to minimize the area. However, area and latency are traded off during datapath merging in [106]. In this chapter, we assume that the designer would use a datapath merging technique more optimized for area as the platform cost is one of our optimization objectives. This merging method can be used by the designer before the DSE to find the area required to implement each task on a dedicated ASIP and the area required for two tasks (i.e. *task pair*) clustered together on an ASIP. These areas can be computed by using area values of hardware components such as an adder, multiplier, etc. that will be used in the final implementation of the ASIP.

During DSE, we estimate the area required by various task clusters and the consequences of datapath merging on the WCET and, hence, on the schedulability of the solution.

We indicate as a_{G_i}, the area of the ASIP required for implementing task τ_i (represented by DFG G_i). The area is expressed in number of gates. The total area required by two merged task τ_i and τ_j (represented by the DFGs G_i and G_j) is given by Equation 8.1, where $a_{i,j}$ is the merged area of two clustered tasks and $a^o_{i,j}$ is the area overhead due to the introduction of *Sel* nodes. The node *Sel* selects one of the inputs and passes it to the output (as shown in Figure 8.1b, in the ASIP implementation corresponds to a multiplexer). Depending on the percentage of dissimilar nodes, the number of *Sel* nodes may vary and it affects the area of the datapath. The area overheads for task pairs can also be computed offline, before DSE. We indicate with N_{Sel}, the number of *Sel* nodes introduced during task merging and with a_{Sel}, the area of a standard *Sel*; hence, the area overhead due to task merging is $N_{Sel} \times a_{Sel}$.

$$a_{G_iG_j} = a_{G_i} + a_{G_j} - a_{i,j} + a^o_{i,j} \tag{8.1}$$

To compute the area requirements of more than two tasks, after the area of the clustered task pair $a_{G_iG_j}$ is computed as shown earlier, the merged/clustered task pair are considered as a new task with area requirements $a_{G_iG_j}$. If another task τ_k represented by DFG G_k is clustered with this new task, then the area of the merged nodes between

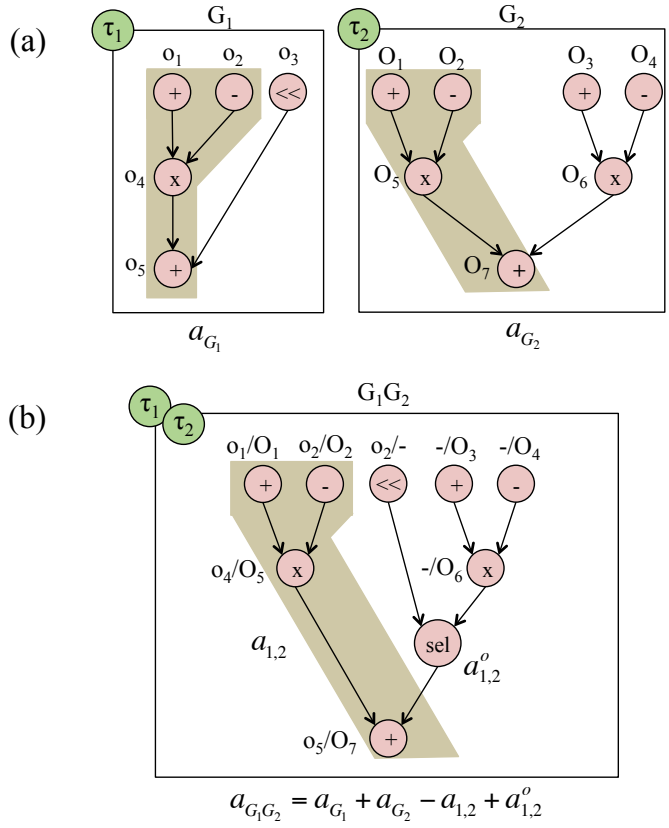


Figure 8.1: A Graph Merging (or Task Clustering) Example, (a) DFG for τ_1 and τ_2 (b) Merged DFG for τ_1 and τ_2 [28]

DFGs G_iG_j and G_k is given by

$$a_{i,j,k} = \min(a_{i,k}, a_{j,k}) \tag{8.2}$$

As shown in Equation 8.2, we consider the merged area between G_iG_j and G_k as the minimum of the area given by similar nodes between tasks in G_i and G_k, and G_j and G_k. We select the minimum area to reduce the area overhead which may also contribute to WCET (discussed in Section 8.2.2): a higher number of merged nodes usually requires a higher number of Sel nodes. The area overhead when DFGs G_iG_j and G_k are merged is

$$a^o_{i,j,k} = \text{Area overhead between } G_k \text{ and } G_{min} \tag{8.3}$$

where G_{min} is the graph among G_i and G_j that has minimum area of similar nodes with G_k. The total area of G_k merged with G_iG_j is given by $a_{G_iG_jG_k} = a_{G_iG_j} + a_{G_k} - a_{i,j,k} + a^o_{i,j,k}$. This area estimation can be iteratively performed until all the tasks in a task cluster are considered. The total area estimated for the task cluster associated to PE_k is indicated as AC_{PE_k}. Our iterative technique for area estimation implies that, given a cluster with multiple tasks, the order in which we consider the tasks for calculating the area will influence the final result. In this chapter, we use a fixed order that is based on the task priority, so that tasks with higher priority incur lesser overhead in the critical path of the DFG (explained in Section 8.2.2). To estimate the area of the entire system, we sum the AC_{PE_k} for every PE in the platform; assuming M PEs in the platform solution, then one of our optimization objective during DSE is to minimize $\sum_{k=1}^{M} AC_{PE_k}$.

8.2.2 Effect of Clustering on the Uncertainty Model

As mentioned in the previous section, datapath merging might influence the WCET, and hence the CDF, of a task due to the introduction of Sel nodes in the critical path of a task, when it is clustered with other tasks. In Figure 8.1b, in the DFG G_2 of task τ_2, there are four critical paths before merging: $O_1 \rightarrow O_5 \rightarrow O_7$, $O_2 \rightarrow O_5 \rightarrow O_7$, $O_3 \rightarrow O_6 \rightarrow O_7$ and $O_4 \rightarrow O_6 \rightarrow O_7$. Due to merging, there is an additional overhead of a *Sel* node in the third and fourth critical paths. The overhead in the WCET originates from the group of nodes that are not merged and feed an input to one of the merged nodes. This can result in increased estimated upper and lower bounds of the CDF. If the WCET overhead for a task during clustering of two tasks is ω cycles (or some unit of time), C^u is the upper bound of the WCET and C^l is the lower bound of WCET, then the CDF for the task is built by shifting the upper and lower bound as $C^u_{new} = C^u + \omega$ and $C^l_{new} = C^l + \omega$, respectively. The WCET overhead for each task pair can be found before DSE when the area overhead is computed and the number of introduced *Sel* nodes is known. Here, we only consider the sharing of resources in the datapath while accounting for the change in the WCET bounds and in the CDF. The sharing of memory and other resources is not considered.

8.2.3 Schedulability Analysis of a task clustering solution

The schedulability analysis with the UM is done using MCS, similarly to what we described Chapter 3. Our approach is to randomly sample, at each MCS iteration, a new value for C_i based on its CDF. Unlike the schedulability analysis described in Chapter 3 where we apply a static non-preemptive scheduling, here we use a fpps.

At each MCS iteration, the sampled C_i value is used to compute the number of tasks

completed within the global deadline d. The schedulability probability is then computed using the number of tasks completed. We call *service* the number of jobs of a task processed by each ASIP within a deadline d. The global deadline that we consider is the hyper-period[1]. The schedulability analysis is performed by computing the worst-case time remaining for each task and, therefore, worst-case service provided to each task. We use the worst-case service as parameter for schedulability analysis because exact service offered to each task is not known until the ASIP architectures are defined. Moreover, instead of using conventional response time analysis (RTA) for schedulability, we use the worst-case service based approach because a large number of applications (such as multimedia applications) would require a certain number of jobs of a task to be completed in a particular time interval instead of having individual job deadlines. Therefore, we first find the worst-case remaining time Δ_i for each task τ_i in accordance to the fpps policy, which is given by:

$$\Delta_i = max \left\{ \left(\Delta_h - \sum_{\tau_j \in hp(\tau_i)} \left(\frac{\Delta_h}{T_j} C_j \right) \right), 0 \right\} \tag{8.4}$$

where $hp(\tau_i)$ is the set of tasks which have higher priority than task τ_i and are clustered together with τ_i on the same ASIP. T_j is the period and C_j is the WCET of the higher priority task τ_j. C_j is one of the Monte Carlo samples from the CDF of τ_j. The hyper-period is denoted by Δ_h. We use Equation 8.4 to compute the worst-case remaining time for a task τ_i in an interval equal to the hyper-period. For this, we first compute the worst-case number of jobs (computed as $\frac{\Delta_h}{T_j}$) of all higher priority tasks and the worst-case times required to process them (computed as $\frac{\Delta_h}{T_j} C_j$). The worst-case processing times of tasks τ_j are then summed up to get the worst-case processing time of the higher priority tasks in $hp(\tau_i)$ and this is finally subtracted from the hyper-period to get the worst-case remaining time for task τ_i. If there is no remaining time after the higher priority tasks are processed, then we set $\Delta_i = 0$. Tasks having the longer critical path are assigned a higher priority [56].

Once we know the worst-case remaining time for task τ_i, we need to verify if this time is enough to service all jobs of τ_i. The maximum number of jobs of a task τ_i that can complete within an interval Δ_h is given by $\frac{\Delta_h}{T_i}$; we compare this value with the number of jobs that can actually complete within the worst-case remaining time Δ_i. If these two values are equal, then the task can be scheduled.

More formally, using Δ_i, we can compute the worst-case number of processed jobs of each task as shown in Equation 8.5, where $t_i = \frac{\Delta_h}{T_i} C_i$ is the worst-case processing time for maximum possible number of jobs of task τ_i in an interval Δ_h. The worst-case num-

[1] least common multiple (LCM) of the periods of the tasks

ber of processed jobs of each task can be used to evaluate the overall schedulability of the task clusters on M ASIPs using the schedulability metric as shown in Equation 8.6.

$$J_i = \begin{cases} \lfloor \frac{\Delta_i}{C_i} \rfloor & \text{if } t_i < \Delta_i \\ \frac{\Delta_h}{T_i} & \text{otherwise} \end{cases} \tag{8.5}$$

$$S_M = \begin{cases} S_M + 1 & \text{if } \sum_i J_i = \sum_i \frac{\Delta_h}{T_i} \\ S_M & \text{otherwise} \end{cases} \tag{8.6}$$

If a task clustering on the ASIPs is schedulable (i.e., all jobs of the tasks mapped to M ASIPs complete within Δ_h or $\sum_i J_i = \sum_i \frac{\Delta_h}{T_i}$), then schedulability metric value S_M is incremented. If the task clustering is not schedulable, S_M does not change. For each iteration of MCS, S_M is updated according to Equation 8.6. To incorporate message communication between tasks on the bus, we check for schedulability of messages on the bus by ensuring that the bus utilization is below 100%. Although this does not guarantee the worst-case end-to-end schedulability (we do not consider the presence constraints between tasks), we provide the task clusters that have a high probability for schedulability of tasks and messages. This can be used as the starting point for ASIP synthesis. Once the ASIPs are synthesized, a detailed end-to-end schedulability analysis can be performed using techniques proposed in [35].

We consider n iterations of MCS corresponding to n sample points of WCET probability density function of each task. Then the probability of schedulability of the platform solution is computed as $p = \frac{S_M}{n}$ if the bus utilization is less than 100%, otherwise p is set to zero. Our optimization objective for performance is to maximize p.

8.3 DSE for Cost and performance optimization

The design space of the possible clustering solutions is huge and cannot be exhaustively explored. Therefore, we use an optimization approach based on a Genetic Algorithm (GA). In particular, among the algorithms described in literature, we use a controlled elitist GA (a variant of the Non-dominated Sorting Genetic Algorithm-II (NSGA-II) [19]) for multi-objective optimization.

GA is a meta-heuristic optimization strategy that defines an initial set of randomly generated candidate solutions called *population*. Each candidate solution is identified by an array called *chromosome*. A set of solutions from the initial population is evolved

through *crossover* and *mutation* that combine and vary the existing chromosomes (parent solutions) to create new child solutions. The child solutions have a better *fitness value* and are used to replace the existing population. This new set of solutions forms the next generation over which the earlier steps are repeated.

The controlled elitist GA that we use allows maintaining some diversity in the population by also retaining solutions with a lower fitness value across different generations. This is important for the convergence to an optimal Pareto front. The GA causes the population to evolve towards a better one until a termination condition is reached. In our case the algorithm stops when there is no improvement in the fitness of the population after a certain number of generations g_{max}.

In a task clustering solution, a *chromosome* is encoded as an array in which each element called *gene* represents a task τ_i and the value assigned indicates the k-th index of PE_k to which the task is assigned. The initial population is composed of Pop solutions. From this initial population, a set of solutions is selected as the parent population. First, the crossover is performed according to a probability P_c. We used a standard single point crossover [15]: given two parents, they are partitioned at a random point and the resulting parts of the two parents solutions are combined to generate a child solution if a randomly generated number $\leq P_c$. Mutation is then applied on the children generated from crossover. A probability P_m is used to determine if each single gene of a child solution is randomly changed or not (i.e. modify the PE associated to each task). After mutation, Pop offspring solutions are generated. Finally, out of the $2 * Pop$ solutions (Pop parent solutions + Pop offspring solutions), the best Pop solutions are selected according to the fitness function. These steps are repeated until a certain number of generations are produced. The parameters g_{max}, Pop, P_c and P_m have been tuned according to the results obtained running multiple executions of the algorithm with different synthetic applications.

The fitness function consists of two optimization objectives discussed in Sections 8.2.1 and 8.2.3: the computation of the *area cost* considering the task similarities and the *schedulability probability* (p). These computations are performed for every candidate platform solution in each generation of GA.

We denote our DSE with task similarities and the UM as Pareto Optimal Clustering method, *Pareto_OM*.

8.4 Experimental evaluation

Without the Pareto_OM approach, the designer would characterize the WCET of each task τ_i with a reference value C_i^{ref} and will not use task similarities to minimize the

area. The optimization objective in this case is the maximization of the number of jobs served considering the C_i^{ref} values of the tasks. We denote this as straightforward method (SFM).

To evaluate the effectiveness of our Pareto_OM approach in deriving a multi-ASIP platform we compared it against the SFM. We used three real-life benchmarks from the Embedded System Synthesis Benchmark Suite (E3S), version 0.9 [2] and four synthetic benchmarks. The real-life benchmarks used from E3S are *consumer-cords*, *telecom-cords* and *networking-cords*. The details of the real-life and synthetic benchmarks are presented in Tables 8.1 and 8.2, respectively. In each of the tables, the number of tasks constituting each benchmark is given in column 2 and the number of ASIPs used for task clustering is given in column 3.

Table 8.1: Results for the realistic case studies [28]

Case Study	No. of Tasks	M	*SFM*		*Pareto_OM*	
			Area	p	Area	p
consumer-cords	12	2	531.011	0%	532.5	74.88%
networking-cords	13	2	182.845	0%	182.845	95.96%
telecom-cords	30	3	1163.929	0%	1148.896	98.16%

Table 8.2: Results for the synthetic case studies [28]

Case Study	No. of Tasks	M	*SFM*		*Pareto_OM*	
			Area	p	Area	p
synth_1	24	4	408.092	0%	394.665	94.04%
synth_2	30	4	521.364	0%	377.344	75.72%
synth_3	34	4	615.237	0%	407.307	55.02%
synth_4	46	6	723.982	0%	493.012	94.22%

The WCET value given in the real-life benchmark is used as the average WCET value C_i^{ref} of task τ_i and the WCET upper (C_i^u) and lower (C_i^l) bounds of each task are obtained by scaling C_i^{ref} with some multiplication factors. For the synthetic benchmarks and for each task, the value of C_i^{ref} is randomly generated and the values of C_i^u and C_i^l are obtained by scaling the corresponding C_i^{ref}.

For real-life benchmarks, the area of each task τ_i (denoted as a_{G_i}) is obtained from the code size of tasks in the E3S benchmark and area of real implementation of a well-

known task (from the benchmark) on a processor. For instance, if the area of the real implementation of a well known task τ_j (such as FFT) obtained from literature is a_j^{real} and the code sizes of tasks τ_i and τ_j are cs_i and cs_j, respectively, the area of the task τ_i was computed as $a_j^{real} \times \frac{cs_i}{cs_j}$. In the case of synthetic benchmarks, these values are generated.

The values for the task similarity and WCET overhead are obtained as follows. Once the area values required for the implementation of each task are obtained, for real-life benchmarks, we estimated the merged area of two clustered tasks (τ_i and τ_j) $a_{i,j}$, the area overhead $a_{i,j}^o$ and the WCET overhead ω by looking at the task similarity. We assumed that a technique as the one presented in Section 8.2.1 is applied. In the case of synthetic benchmarks, $a_{i,j}$ and $a_{i,j}^o$ are generated such that they did not exceed the area of each task, while the WCET overhead ω is manually generated such that the WCET overhead did not exceed C_i^{ref} value of each task in a clustered task pair.

The parameters of GA were tuned as follows. The initial population size was set to $Pop = 100$. The crossover and mutation probability were set to $P_c = 0.4$ and $P_m = 0.2$, respectively. The GA terminated when there was no improvement in the fitness function for 6 generations. The maximum number of generations were set to $g_{max} = 100$. The execution time of the DSE for the benchmarks varied between 10 minutes to 1 hour.

In our experiments, we presented the advantage of our proposed platform synthesis approach Pareto_OM in comparison to the SFM approach for real-life benchmarks and synthetic benchmarks. In order to compare the performances of the two approaches, the task clustering solution obtained by SFM was then evaluated under the inclusion of WCET uncertainty and task similarity effects.

The results obtained using both approaches are presented in Table 8.1 and Table 8.2. We provided the area cost (in KGates) and schedulability probability, p, obtained with the SFM approach in columns 4 and 5. The same results obtained with Pareto_OM approach are shown in columns 6 and 7.

The Pareto_OM approach generates a Pareto front of solutions. We only report the solution that gives maximum schedulability probability, but still has lesser area than what is obtained using SFM in Table 8.1 and Table 8.2. These are the task clustering solutions that should be considered for the platform synthesis. It is clear from the results that the solution obtained using Pareto_OM approach outperforms the solution obtained using SFM approach for all the real-life and synthetic benchmarks.

We also present the Pareto plots for the real-life and synthetic benchmarks. For *consumer-cords* (Figure 8.2a), using the Pareto_OM approach, there are two solutions which have a high schedulability probability, but require higher area in comparison to the

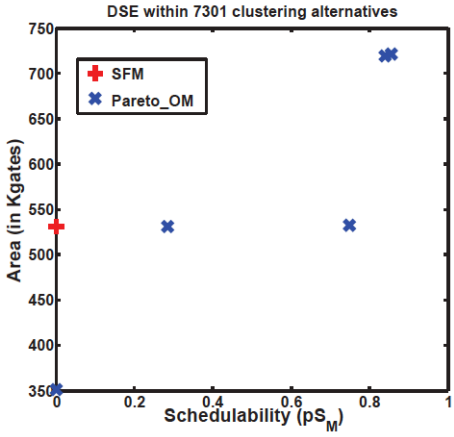
SFM approach. Therefore, a better choice would be to select the solution given in Table 8.1. The comparison between the SFM approach and the Pareto_OM approach for *networking-cords* is shown in Figure 8.2b. Although the Pareto_OM approach returns a couple of solutions with almost comparable area cost to the solution given by SFM approach, these solutions have a higher probability of schedulability. This result highlights the fact that for *networking-cords*, the solution that optimizes for performance is not able to exploit the task similarities well to reduce area. The comparison between the SFM approach and the Pareto_OM approach for *telecom-cords* is shown in Figure 8.2c. In this case, there a few solutions proposed by Pareto_OM that have higher probability of schedulability in comparison to the solution proposed by SFM with significant area savings. This result is because the task similarities in *telecom-cords* are well exploited by the Pareto_OM approach. However, there are fewer points on the Pareto front because the message schedulability condition (Section 8.2.3) on the bus was not satisfied.

The comparison of SFM approach and the Pareto_OM approach for the synthetic benchmarks confirms the results obtained for the real-life benchmarks. *Synth_1* is shown in Figure 8.3a; our Pareto_OM approach produces clustering solutions with better area cost and performance in comparison to SFM approach. Figure 8.3b shows the comparison of SFM approach and the Pareto_OM approach for *synth_2*. There are multiple clustering solutions shown for *synth_2*, which save considerable area in comparison to SFM approach, but still having a higher p in comparison to the clustering solution proposed by SFM approach in red marker. Better clustering solutions were also observed for *synth_3* (Figure 8.3c) and *synth_4* (Figure 8.3d) using the Pareto_OM approach. For *synth_3*, the schedulability probability is lower than the other synthetic benchmarks because the contention on the bus due to inter-processor task communication does not satisfy the message schedulability condition.

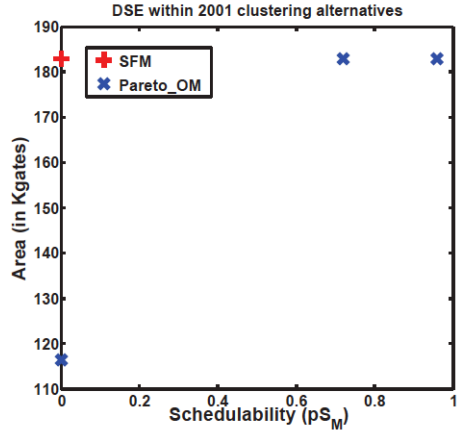
8.5 Summary

In this chapter, we proposed an extension of our DSE with UM for multi-ASIP platform synthesis. We applied the UM together with a fpps scheduling to evaluate the schedulability probability of multiple task clustering solutions. We also introduced a method to evaluate the area cost, based on datapath merging techniques and task similarities. An NSGA-II optimization algorithm was used for running the DSE. The efficacy of our approach was demonstrated using real-life and synthetic benchmarks. From the experimental results, we observed that our proposed approach (Pareto_OM) could find platform solutions that exhibited lesser cost and higher schedulability probability in comparison to the SFM approach.

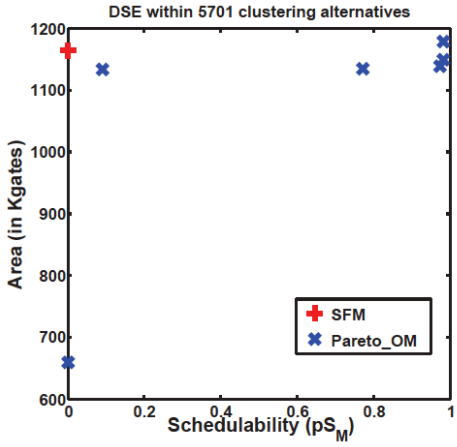
Additionally, in this chapter, we demonstrated that our UM can be applied to different



(a) Comparison of results using SFM and Pareto_OM for *consumer-cords* case study



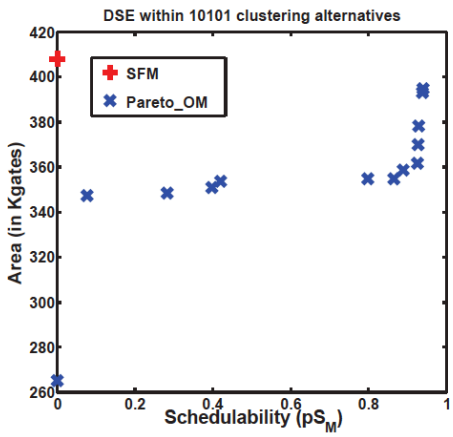
(b) Comparison of results using SFM and Pareto_OM for *networking-cords* case study



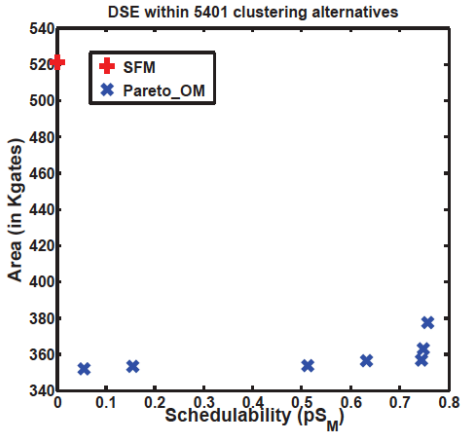
(c) Comparison of results using SFM and Pareto_OM for *telecom-cords* case study

Figure 8.2: Results obtained for the real-life benchmarks [28]

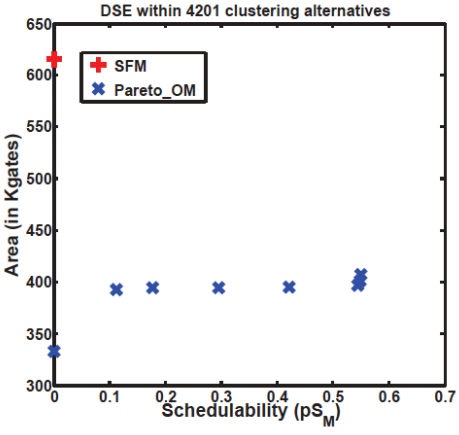
schedulability analysis and, therefore, can have a general application targeting also multiple types of ASIP architectures. In the previous chapters, we considered a static schedulability analysis, while in this chapter, we applied a dynamic and preemptive scheduling.



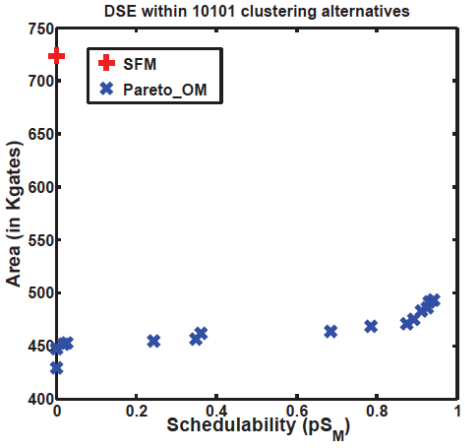
(a) Comparison of results using SFM and Pareto_OM for synth_1 case study



(b) Comparison of results using SFM and Pareto_OM for synth_2 case study



(c) Comparison of results using SFM and Pareto_OM for synth_3 case study



(d) Comparison of results using SFM and Pareto_OM for synth_4 case study

Figure 8.3: Results obtained for the synthetic benchmarks [28]

CHAPTER 9

Conclusion

This final chapter consists of two sections: Section 9.1 contains a summary of the contributions and additional comments on the work described in this thesis and Section 9.2 describes the open issues and directions for future research.

9.1 Contributions

In this thesis we presented an innovative approach for the system-level design of multi-ASIP platforms optimized to run one or more input applications. We targeted streaming applications that we modeled initially with a task graph and then with a SDFG to take advantage of the task level and pipeline level parallelism. We introduced a methodology that is applied in the very early phases of the design to support the designer in identifying the platform composition. We believe in the novelty of our approach and, to the best of our knowledge, there are no other methods in the literature that address the *circular dependencies* described in Section 1.3. The circular dependency derives from the fact that the micro-architecture of an ASIP is designed according to the tasks that it has to run (i.e. task clustering); at the same time, to evaluate a task clustering solution, we need the WCET of each task, which is available only when the micro-architecture of the ASIP is defined. Our solution breaks this circular dependency by using a probabilistic approach called UM.

The UM is used to estimate the performance of a task running on an ASIP which has not been designed yet. For each task, our UM predicts the range of performance (i.e. WCET) of a set of possible micro-architecture configurations. The range of performance is limited by two values, an upper and lower bound. In Chapter 2, we demonstrated that a normal distribution, drawn between these two bounds, is a valid approximation of the probability distribution of the WCETs of a task when executed on multiple micro-architecture configurations.

Given as input the application model, an initial platform model (that specifies the number of PEs and how they are interconnected) and the UM for each task, we build a static non-preemptive schedulability analysis for the evaluation of a task clustering solution (Chapters 3 and 5). Our evaluation is implemented using Monte Carlo simulation and considers the data dependencies between the tasks. The data dependencies are modeled as messages that are assigned to the communication architecture. Also the resource (PEs and CEs) contention is considered. The output of our schedulability analysis is a task clustering solution, with the highest chance of meeting the application deadline after the platform implementation. A task clustering solution indicates how the tasks are assigned to the different PEs, which in turns defines, how many ASIPs we should include in the platform and which bus type should be used to interconnect them.

For the applications modeled as a SDFG, we split the schedulability analysis into two parts, one that evaluates the scheduling of a single iteration of the SDFG (TLA) and one for the estimation of multiple iteration of the SDFG considering the pipeline parallelism at macro-architecture level (PA). This speeds up the schedulability analysis of a task clustering solution and allows reducing the execution time of the SSEA DSE used to explore the design space of the task clustering solutions.

Our DSE with UM, applied to the task graph and SDFG models, is evaluated using multiple case studies, including real applications as MJPEG, ECG and SC. For the real case studies, the evaluation is done implementing the multi-ASIP platform associated to the best task clustering solution with SH tools and technology (Chapters 4 and 6).

Additionally, we evaluated the influence of the upper and lower bounds used to define the UM of each task. Our analysis suggests that is not necessary to define these values with high accuracy; the code analysis tool that we used for the estimation of the upper and lower bounds can produce relatively high estimation errors, depending on the task and the application. For example, for the MJPEG encoder modeled as a SDFG, there is an error up to 41%, however, even with such relevant errors, our DSE is able to identify the proper task clustering. Our intuition is that it suffices to have accuracy in the relative number of execution cycles of the tasks when comparing them to the execution cycles of the entire application.

In Chapter 7, we presented how our DSE with UM is integrated into the ASAM flow. In relation to ASAM, we described the two tools for macro-architecture DSE that we

developed. A *probabilistic DSE* tool implementing the UM that is used at the beginning of the design flow when there is no information about the micro-architecture of each ASIP, and a *deterministic DSE* tool that is used in a later phase of the design when, for each task cluster identified, we have a set of possible micro-architecture configurations. Using this more refined information, the *deterministic DSE* runs a multi-objective DSE optimizing area and performance of the entire platform. For the integration of the tools with the ASAM flow, we generated ad-hoc XML interfaces.

Finally, we presented an extension to our DSE with UM. Up to Chapter 7, we considered a static non-preemptive schedulability analysis that can be implemented by SH ASIPs. On the other hand, in Chapter 8, we show that the UM can be applied to different scheduling policies, in particular we applied a fixed priority preemptive scheduling. We also proposed a method for evaluating task similarities. We used it to cluster together tasks that share a similar datapath with a consequent minimization of the micro-architecture area. We implemented a multi-objective DSE, which objectives are the maximization of the schedulability probability and the minimization of the area.

We believe that the UM described in this thesis can help designers to speed up the definition of a multi-ASIP platform by automatically evaluating multiple task clustering solutions. Our DSE required around thirty minutes for the more complex case studies and a few minutes for the smaller ones.

Moreover, our intent is to propose a general approach for the design of a multi-ASIP platform. However, knowing the characteristics of the ASIP technology used, it is possible to improve the results produced by our estimation. In Chapters 4 and 6, we described how we improved our schedulability analysis including some aspects of the underlying SH ASIP technology. For example, we adjusted the bus model to be consistent with the SH one and we considered the extra cycles required by the synchronization between tasks assigned to different clusters. Additionally, in our schedulability analysis (Chapters 3 and 5), we considered a static non-preemptive scheduling policy that can be implemented with the single threaded VLIW ASIPs from SH. Other types of architecture might allow using the UM with other schedulability analysis like the fpps policy described in Chapter 8.

9.2 Open issues

There is still a number of open issues related to the application of the UM that is worth exploring. Below we will list some of them.

During the DSE with UM we used a bus-based platform as communication architecture, with the option of evaluating different types of buses. We also described a small exam-

ple in which the DSE with UM is applied to an initial platform model with a MESH NoC. Depending on the applications in input and the number of ASIPs in the platform, it becomes important to explore more intensively the impact of different communication architectures that can be included during DSE. An option would be to start the DSE with a single bus, then, switching to a NoC implementation in case the communication architecture is identified as the bottleneck of the performance. We can use the PA described in the schedulability analysis with a SDFG application model. With PA, we can calculate the size of the slowest pipeline stage that is the one determining the total performance of the platform. A pipeline stage corresponds to a PE or CE. If the slowest pipeline stage is due to the communication network, it is possible to substitute the bus with a NoC.

Another aspect to investigate is the inclusion of the area estimation to the DSE with UM used in the ASAM design flow. The task similarities approach described in Chapter 8 can be applied here. Another option would be to use the information that can be extracted by the code analysis tool: while estimating the lower bound value, that is the expected number of cycles when the task is executed on a virtual processor according to an ASAP scheduling without architectural constraints, it is also possible to calculate the maximum ILP of each task. Moreover, it is also straightforward to get the ILP per instruction type. This value can be used to determine the maximum number of resources (e.g., FUs) required by each task. The similarity in the level of parallelism of the tasks can be used to determine which tasks should be clustered together. Furthermore, also the size of the local memories of each ASIP can be optimized during the macro-architecture DSE. At system-level it is possible to consider the size of the messages exchanged by the tasks and evaluate how the data exchange influences the sizes of the local memories.

In this thesis, we presented and motivated the UM and we applied it to different application models and schedulability analysis. The results obtained show that associating the UM with a schedulability analysis can guide the macro-architecture DSE for a multi-ASIP platform when there is no pre-established information about the underlying hardware implementation.

APPENDIX A

Additional results

A.1 Additional results from Experiment 2

In this section, there are the additional results obtained for Experiment 2 from Section 2.3.1.2. Figures A.1, A.2, A.3 and A.4 contain the relative error between the scheduling length calculated usign normal, Gumbel and uniform distributions and the scheduling length obtained using a deterministic DSE, DSE_{det}. The four figures refer to the case studies in Table 2.3.

A.2 Additional case studies using SDFG application model

In this section, we present the details of the ECG and Spatial Coding (SC) case studies described in Chapter 6. The results presented in this section are partially described in [70] that is currently under revision.

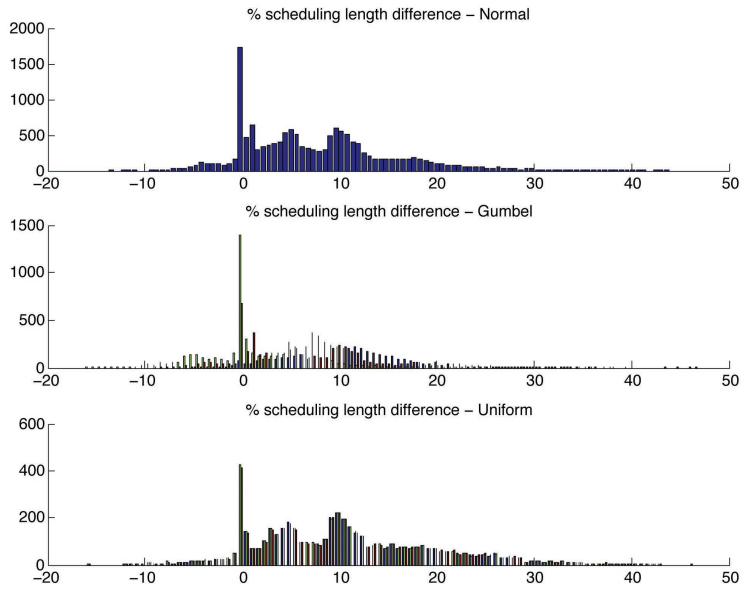


Figure A.1: Histogram of the percentage (%) differences in the scheduling length for Case Study 2

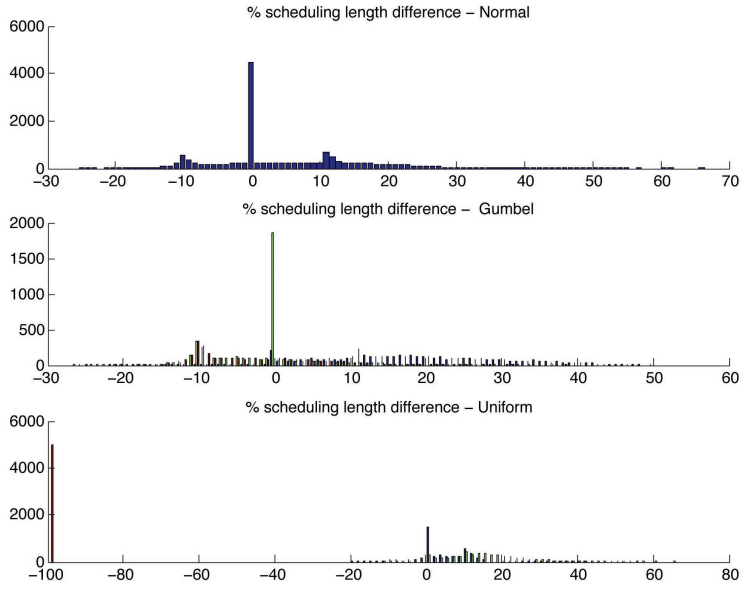


Figure A.2: Histogram of the percentage (%) differences in the scheduling length for Case Study 3

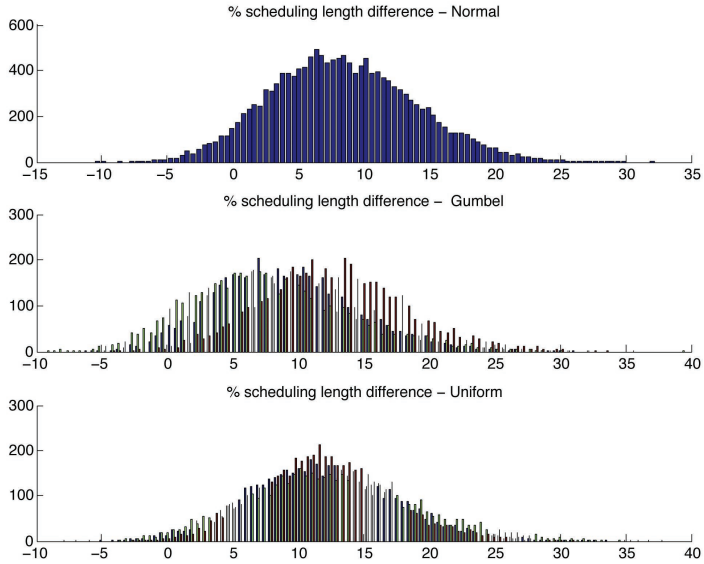


Figure A.3: Histogram of the percentage (%) differences in the scheduling length for Case Study 5

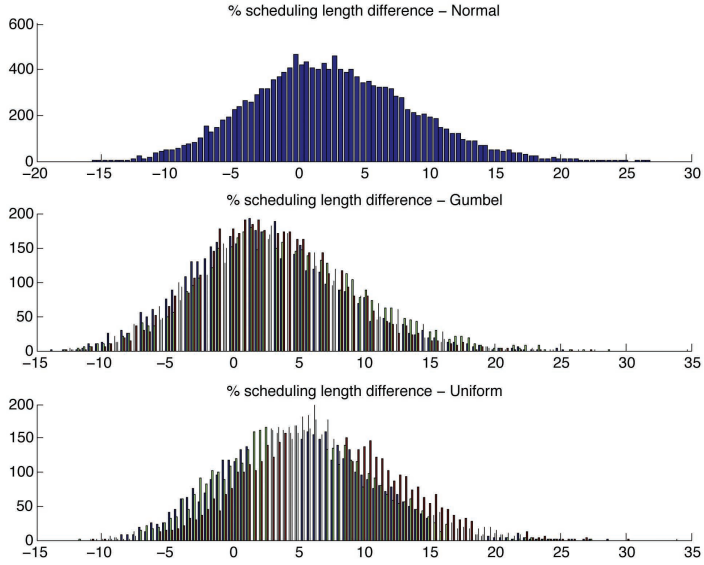


Figure A.4: Histogram of the percentage (%) differences in the scheduling length for Case Study 6

A.2.1 ECG case study

In this section, we present the ECG application: we model the application as a SDFG and we apply our DSE with UM to it. The KPN and the SDFG of the ECG application are shown in Figure A.5a and A.5b. In the KPN of ECG, between actors ND_7 and ND_8 there are multiple edges. From the C code analysis, we can determine that we fall under *case B* (as presented in Section 4.2), i.e., all messages are executed when ND_7 completes. We generate a message between ND_7 and ND_8 with size equal to the sum of the data to transfer on all edges.

In Table A.1 (first row), there are the input constraints for the ECG case study: we consider the elaboration of 10,000 ECG samples measured with a sampling frequency of 600 Hz, giving a deadline of $16s$. We suppose a frequency f of $1MHz$ and $PC_{max} = 2$. The average numbers of cycles for each task firing (obtained using the code analysis tool described in the ASIP DSE (Phase 1) of [43]) are in Table A.2.

We built the CDF for each task using the estimated C^l and C^u and the input frequency $f = 1MHz$ (Figure A.6). The amount of data expressed in bits of each message is shown in Table A.3. Then we ran our DSE (for $200s$ and with $n = 5,000$): the best clustering solution found has a probability $p_{ECG} = 0.56$ (Figure A.7) and uses two ASIPs. The task clustering and its cost are summarized in the first row of Table A.4 (Sol_1). We generated the ASIPs according to the task clustering solution found and using the micro-architecture design tool described in [43]. We obtained three-issue slot ASIPs; the final platform contains the same interconnections as the platform implemented for the MJPEG encoder (Figure 4.5b).

Then we implemented the cores and the platform using SH tools and we ran the ECG code obtaining a schedulable solution. Columns 5-6 of Table A.4 show the number of cycles and the execution time (at a frequency $f = 1MHz$) that we obtained with SH simulator.

In Table A.4 and Figure A.7, there are the results that we obtained for additional task clustering solutions (Sol_2, Sol_3, Sol_4 and Sol_5). We verified that our UM was able to identify which clustering solutions were better previous the actual implementation of the multi-ASIP platform. There was a correspondence between the probability of meeting the deadline (and the value of the quantile function) that we estimated and the actual schedule length that we obtained from the SH simulator: to a higher probability corresponded also a shorter schedule length. All solutions are schedulable except for Sol_5.

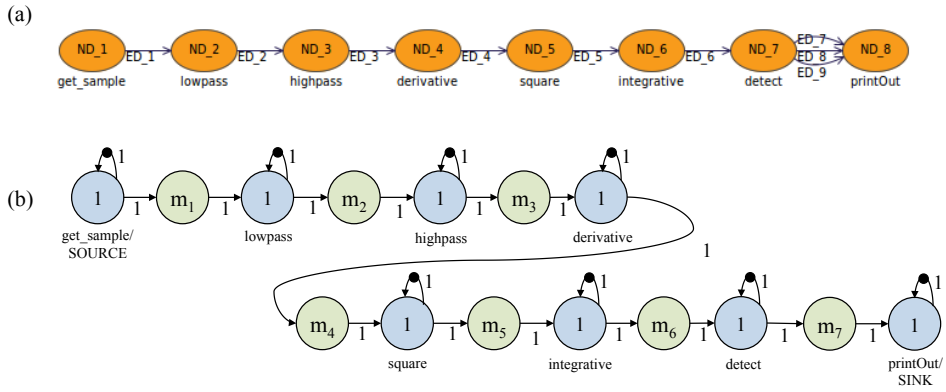


Figure A.5: KPN model generated by Compaan for the ECG (a) and corresponding SDFG model (b)

Table A.1: Input constraints for ECG and SC applications

Case study	$d\ (\mu s)$	$f\ (MHz)$	PC_{max}	*Bus type*
ECG	16,000,000	1	2	b_{32}^{1}
SC	205,000	1,600	3	$b_{32}^{1,600}$

Table A.2: C values for ECG (average number of cycles for a single iteration of the task)

C	$lowpass$	$highpass$	$derivative$	$square$	$integrative$	$detect$
C_j^u	46	85	1	1	2038	16
C_j^l	41	46	1	1	1084	10

Table A.3: Message sizes (in bits) for ECG

m_1	m_2	m_3	m_4	m_5	m_6	m_7
32	32	32	32	32	32	96

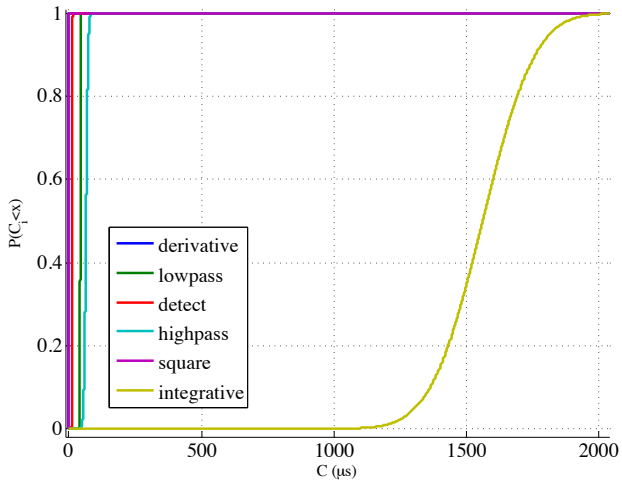


Figure A.6: Cumulative distribution functions for the tasks of the ECG application (with $f = 1MHz$)

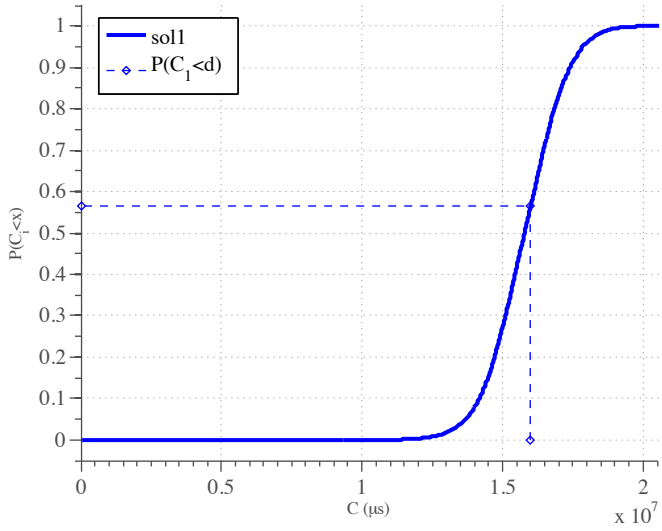


Figure A.7: Results from the macro-architecture DSE for ECG

Table A.4: Comparison of clustering solutions for ECG

Sol_{ID}	Clusters		$P(\delta_{A_{ECG}} < d_{ECG}^{10,000})$	$\delta_{A_{ECG}}^{0.5} = P^{-1}(p_{0.5})$ (μs)	*sim* (cycles)	*sim* (μs)	sched
	PE_1	PE_2					
1	lowpass, high-pass, deriva-tive, square	integral, detect	0.56	15783000	13790796	13790796	yes
2	lowpass, high-pass, deriva-tive	square, integral, detect	0.54	15853000	14000776	14000776	yes
3	lowpass, high-pass	derivative, square, integral, detect	0.52	15928000	14460733	14460733	yes
4	lowpass, high-pass, deriva-tive, square, integral	detect	0.23	16942800	15934010	15934010	yes
5	lowpass, high-pass, deriva-tive, square, inte-gral, detect	-	0.22	16991000	16692594	16692594	no

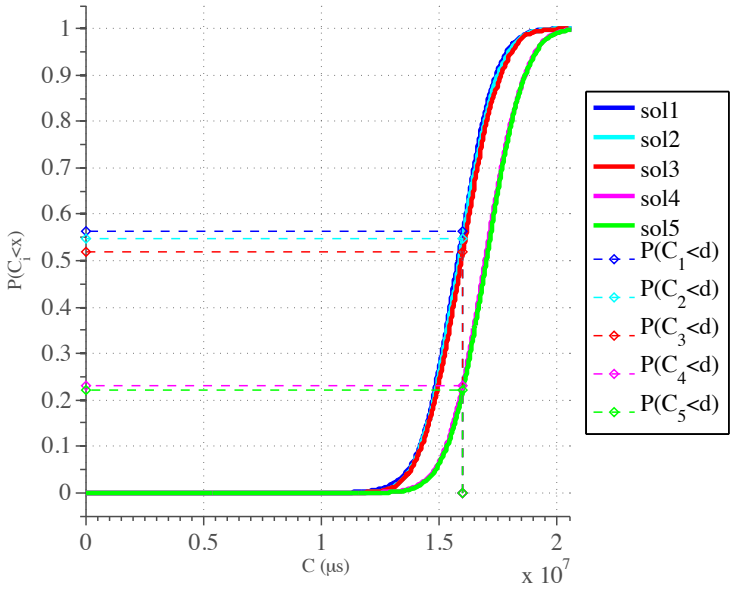


Figure A.8: Comparison of the CDF of different clustering solutions for ECG

Table A.5: Comparison between the number of cycles estimated by the profiling tool [43] and the ones obtained from simulation for ECG

Task Name	C_j^l (cycles)	C_j^u (cycles)	sim (cycles)	$\%Tot_{C_j^l}$	$\%Tot_{sim}$	Err $\lvert\%Tot_{C_j^l}$ - $\%Tot_{sim}\rvert$
lowpass	400,015	450,010	750,016	3.39	4.55	1.16
highpass	450,084	840,021	1,330,203	3.81	8.08	4.26
derivative	10,000	10,000	350,012	0.08	2.12	2.04
square	10,000	10,000	100,000	0.08	0.61	0.52
integrative	10,830,071	20,370,915	12,753,683	91.78	77.42	14.36
detect	99,614	155,070	358,661	0.84	2.18	1.33
Total (cycles)	**11,799,784**	**21,836,016**	**16,472,592**			

A.2.2 SC case study

In this section, we present the Spatial Coding application (part of MPEG4) provided by STMicroelectronics [91]. The KPN and the SDFG of the SC application are depicted in Figures A.9 and A.10, respectively. The multiple edges between the same couple of nodes in the KPN in Figure A.9 fall under *case B* and they are substituted in the SDFG by a single message with size equal to the sum of the data to transfer on all edges.

In Table A.1 (second row), there are the input constraints for the SC case study: we consider the elaboration of 5 frames, each of them composed of 40x30 blocks. This give us $iter = 6,000$ for the SDFG of SC. We have $f = 1,600MHz$ and $PC_{max} = 3$[1]. As we are considering a throughput of 24 fps, we have a deadline $d_{SC}^{6,000} = 205,000\mu s$.

The average number of cycles for each task firing (obtained using the code analysis tool in [43]) is in Table A.6. Some rows in the table contain multiple tasks (e.g., fefoIDCT8_{1,2} indicates the two tasks fefoIDCT8_1 and fefoIDCT8_2): tasks with a similar naming have the same C code; therefore, they also have the same estimated upper and lower bound values (if depending on the data in input, they also have the same profiled execution).

With the values in Table A.6, we built the CDF for each task using the estimated C^l and C^u. The resulting CDFs are represented in Figure A.11 (due to the very dissimilar sizes of the tasks we split them in two figures). The size in bits of each message of the SDFG is shown in Table A.7.

The design space for the SC case study is bigger than the one of the MJPEG and ECG; hence we ran our macro-architecture DSE for $1,800s$. We used $n = 5,000$. Our DSE found a task clustering solution with $p_{SC} = 0.99$ and with three ASIPs. The details of the solution are summarized in the first row, Sol_1, of Table A.8. The CDF of Sol_1 is in Figure A.13. For the SC case study, we manually defined the micro-architecture of the three processors (due to some incompatibility of the code representation available with the tool described in [43]). We used the same library of IS used for the MJPEG and ECG case study and we implemented three processors, PE_1 and PE_3 with 4 ISs and PE_2 with 3 ISs (including the default IS). Then we built the three-ASIP platform (Figure A.12): we needed synchronization FIFOs between PE_1 and PE_2 and between PE_2 and PE_3 (there was no direct communication between PE_1 and PE_3 so we could avoid the insertion of extra FIFOs) and one FIFO adapter for each ASIP (for synchronization between host and ASIPs).

We clustered the tasks to the three ASIPs as suggested by our DSE and we obtained a

[1] We set the frequency to 1,600 MHz to find a schedulable solution after the implementation of the platform; however, we are aware that it is not a realistic frequency and the optimization of the application code and additional processors should be used to achieved the desired performance at a lower frequency.

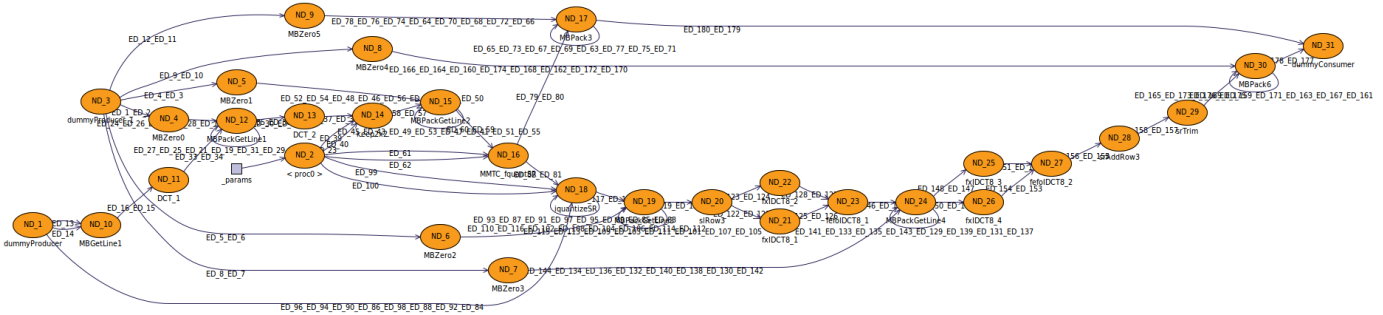


Figure A.9: KPN model generated by Compaan for Spatial Coding case study

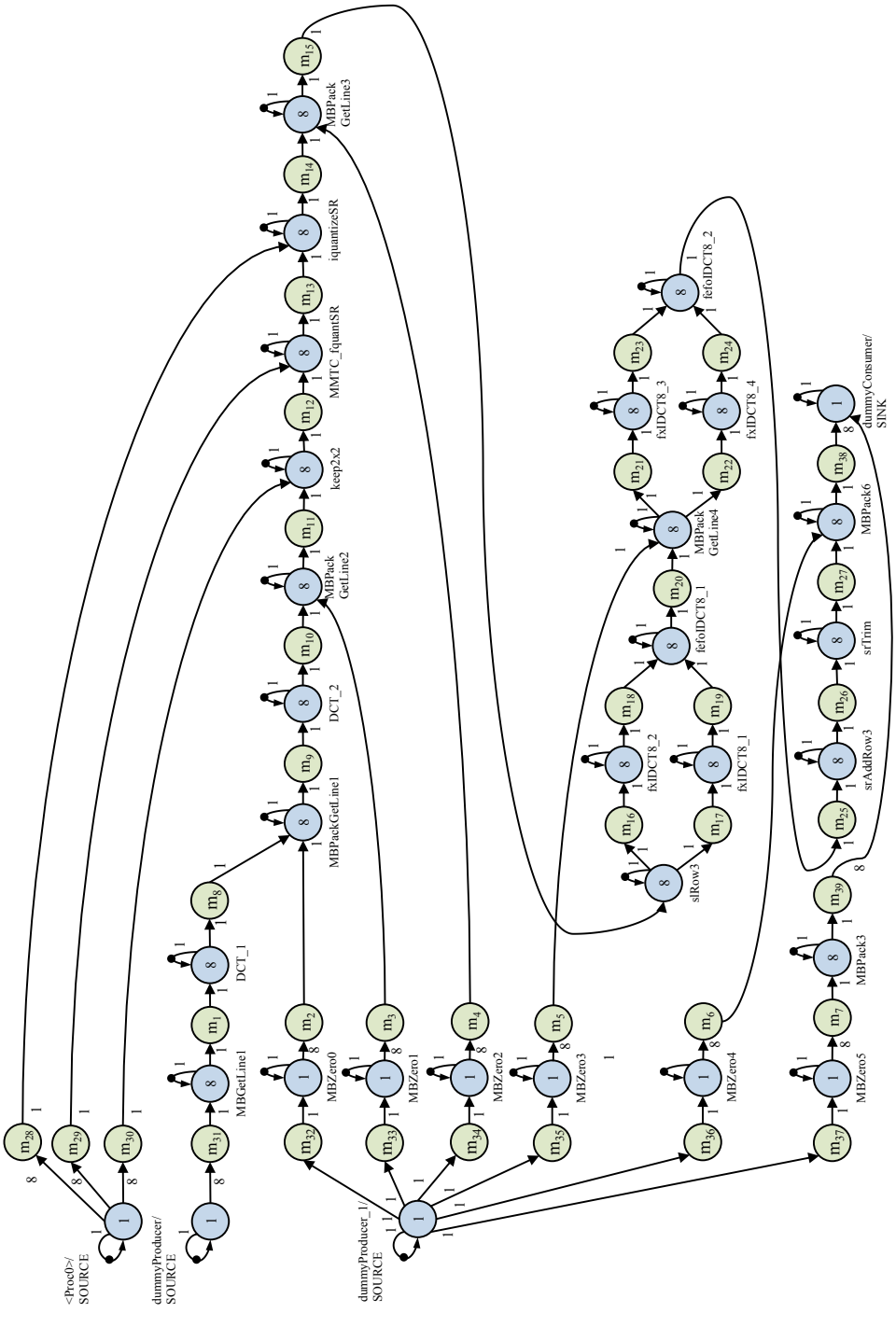


Figure A.10: SDFG model for Spatial Coding case study

Table A.6: C values for SC (average number of cycles for a single iteration of the task)

Task Name	C_j^l	C_j^u
MBPackGetLine3	53	77
fxIDCT8_{1,2,3,4}	160	467
keep2x2	16	19
srAddRow3	16	35
srTrim	122	146
iquantizeSR	227	314
MBPack3	31	47
MBPackGetLine{2,4}	38	54
MBPack6	16	24
slRow3	16	26
fefoIDCT8_{1,2}	16	38
MMTC_fquantSR	4929	6520
MBPackGetLine1	44	61
MBZero{0,1,2,3,4,5}	1	2
DCT_{1,2}	70	215
MBGetLine1	27	36

Table A.7: Message sizes (in bits) for SC

$m_2 - m_7$, $m_{31} - m_{39}$	$m_{28} - m_{30}$	m_{18}-m_{19}, m_{23}-m_{24}	$m_1, m_8 - m_{17}$, $m_{20} - m_{22}$, m_{25}-m_{27}
2048	224	128	256

schedulable solution that run for 274,847,324 cycles (171,799.58 μs at $f = 1,600MHz$).

As for the previous case studies, to demonstrate the validity of the solution found, we compared it with other task clustering solutions that are summarized in Table A.8: Sol_2 has a single ASIP, while Sol_3, Sol_4 and Sol_5 use three ASIPs. Please note that depending on the clustering solutions, the ASIPs and their interconnections may change. Except for Sol_2 that had a probability ~ 0 to meet the deadline, the other task clustering solutions, once implemented, provided schedulable implementations. Due to the high number of tasks in the SC application, it was possible to select multiple task clustering solutions to compare with. We selected Sol_2 to analyze a solution in which task

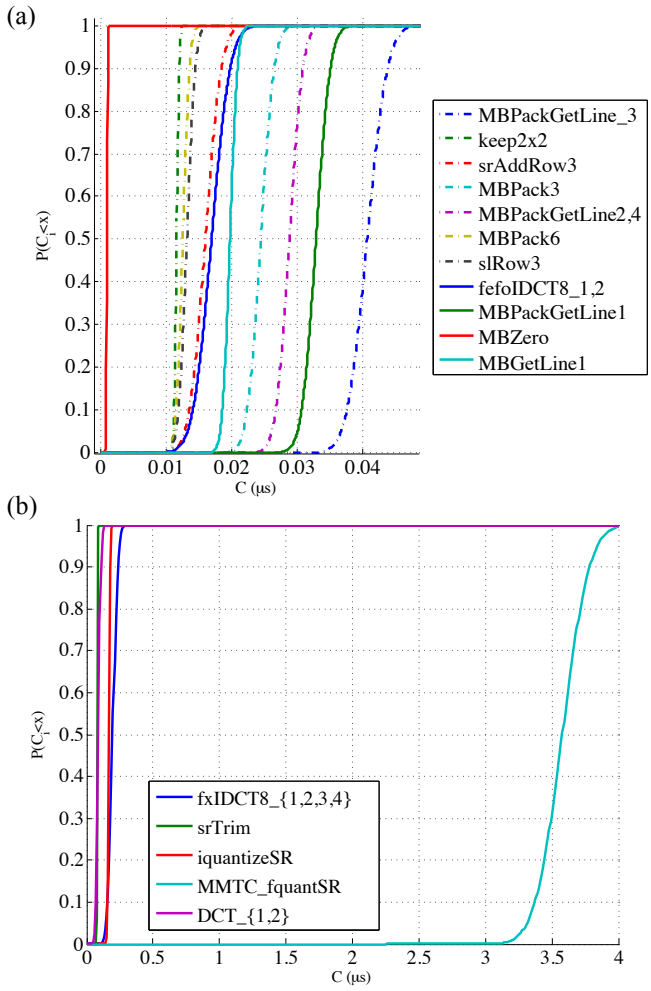


Figure A.11: (a) and (b) Cumulative distribution functions of the tasks of the SC application (with $f = 1,600MHz$)

level and pipeline parallelism (at system level) could not be exploited and to get an estimation of the performance obtainable on a single ASIP. We selected Sol_3 and Sol_4 as they did not differ very much from the task clustering found by our DSE: we evaluated those solutions in which the pipeline parallelism could be conveniently exploited and in which the communication and synchronization between processors were not the bottleneck. In particular, analyzing Sol_3, we verified that it is not convenient to cluster tasks with MMTC_fquantSR. Analyzing Figure A.11 with the CDFs of the single

tasks, we observed that task MMTC_fquantSR had the highest C^l and C^u; therefore, clustering it with other tasks penalized the pipeline parallelism (even if MBZero4 is a task with almost negligible WCET). When we evaluated Sol_3 and Sol_4 with our UM, we obtained a probability of 0.99, but we could observe higher values in the quantile function $\delta^{0.5}_{A_{SC}}$; these results are reflected in higher scheduling length. In Sol_5 we explored a clustering solution in which the task level parallelism between tasks fxIDCT_1 and fxIDCT_2 can be exploited. In this case we have a lower probability (0.95) and a higher scheduling length than the previous solutions after implementation.

In this section we demonstrated that our UM and the associated DSE can explore in a reasonable time (less than 1 hour) multiple task clustering solutions and provide a good indication of which task clustering should be selected. Even if we cannot claim to find a schedulable task clustering solution with our UM, we demonstrated that we could find a clustering solution that has a high probability of being schedulable after design. Additionally, we showed that the probability and the quantile function values can be used to compare different task clustering solutions and that the results obtained after implementation are consistent with our evaluations. Our approach can offer a valid starting point for a designer that has to implement a multi-ASIP platform in which the ASIPs have not being designed yet.

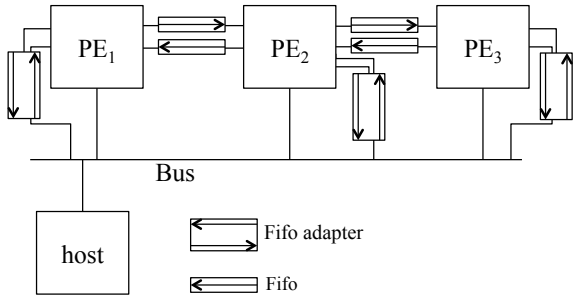


Figure A.12: Block schematic of the platform generated for Sol_1 for SC

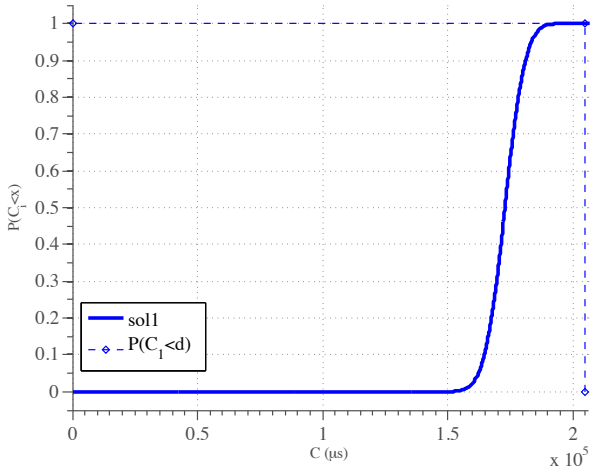


Figure A.13: Results from the macro-architecture DSE for SC

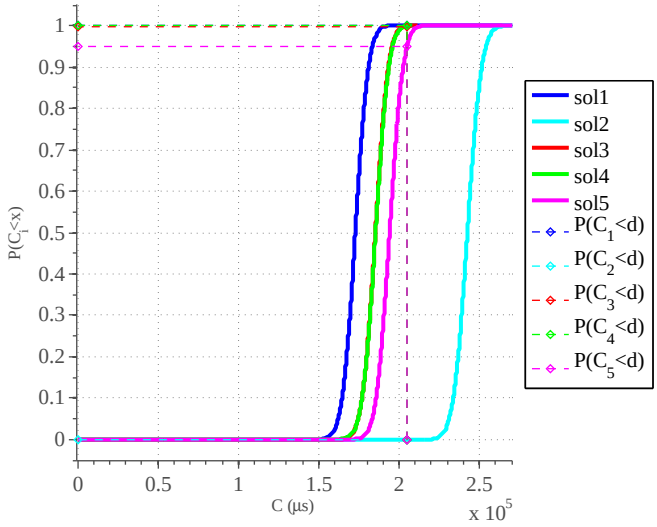


Figure A.14: Comparison of the CDF of different clustering solutions for SC

Table A.8: Comparison of clustering solutions for SC

Sol_{ID}	Clusters			$P(\delta_{A_{SC}} < d_{SC}^{6000})$	$\delta_{A_{SC}}^{0.5} = P^{-1}(p_{0.5})\ (\mu s)$	*sim* (cycles)	*sim* (μs)	sched
	PE_1	PE_2	PE_3					
1	MBGetLine1, DCT_{1,2}, MBZero{0,1,2,3,4,5}, MBPackGetLine{1,2}, keep2x2	MMTC_fquantSR	iquantizeSR, MBPackGetLine{3,4}, slRow3, fxIDCT8_{1,2,3,4}, fefoIDCT8_{1,2}, srTrim, srAddRow3, MBPack6, MBPack3	0.99	172000	274847324	171799.58	yes
2	MBGetLine1, DCT_{1,2}, MBZero{0,1,2,3,4,5}, MBPackGetLine{1,2}, keep2x2, MMTC_fquantSR, iquantizeSR, MBPackGetLine{3,4}, slRow3, fxIDCT8_{1,2,3,4}, fefoIDCT8_{1,2}, srTrim, srAddRow3, MBPack6, MBPack3	-	-	0	242600	573715348	358572.09	no
3	MBGetLine1, DCT_{1,2}, MBZero{0,1,2,3,5}, MBPackGetLine{1,2}, keep2x2	MBZero4, MMTC_fquantSR	iquantizeSR, MBPackGetLine{3,4}, slRow3, fxIDCT8_{1,2,3,4}, fefoIDCT8_{1,2}, srTrim, srAddRow3, MBPack6, MBPack3	0.99	185400	275255614	172034.76	yes
4	MBGetLine1, DCT_{1,2}, MBZero{0,1,2,3,4,5}, MBPackGetLine{1,2}, keep2x2, iquantizeSR	MMTC_fquantSR	MBPackGetLine{3,4}, slRow3, fxIDCT8_{1,2,3,4}, fefoIDCT8_1,2, srTrim, srAddRow3, MBPack6, MBPack3	0.99	185500	278939568	174337.23	yes
5	MBGetLine1, DCT_{1,2}, MBZero{0,1,2,3,4,5}, MBPackGetLine{1,2}, keep2x2	MMTC_fquantSR, iquantizeSR, MBPackGetLine3, slRow3, fxIDCT8_1	MBPackGetLine4, slRow3, fxIDCT8_{2,3,4}, fefoIDCT8_{1,2}, srTrim, srAddRow3, MBPack6, MBPack3	0.95	194300	303402927	189626.83	yes

Table A.9: Comparison between the number of cycles estimated by the profiling tool [43] and the ones obtained from simulation for SC

Task Name	C_j^l (cycles)	C_j^u (cycles)	sim (cycles)	$\%Tot_{C_j^l}$	$\%Tot_{sim}$	Err $\lvert\%Tot_{C_j^l}$ - $\%Tot_{sim}\rvert$
MBPackGetLine3	2,556,000	3,708,000	16932000	0.83	2.95	2.12
fxIDCT8_{1,2,3,4}	7,680,000	22,416,000	23,688,000	2.50	4.13	1.62
keep2x2	792,000	936,000	8,040,000	0.26	1.40	1.14
srAddRow3	768,000	1,680,000	2,496,000	0.25	0.44	0.18
srTrim	5,856,000	7,008,000	6,240,022	1.91	1.09	0.82
iquantizeSR	10,908,003	15,090,000	37,308,000	3.56	6.51	2.95
MBPack3	1,500,000	2,268,000	12,720,000	0.49	2.22	1.73
MBPackGetLine{2,4}	1,806,000	2,574,000	17,700,000	0.59	3.09	2.50
MBPack6	750,000	1,134,000	12,720,000	0.24	2.21	1.97
slRow3	768,000	1,248,000	2,016,000	0.25	0.35	0.10
fefoIDCT8_{1,2}	768,000	1,824,000	3,216,000	0.25	0.56	0.31
MMTC_fquantSR	236,579,118	312,956,828	275,885,112	77.17	48.10	29.06
MBPackGetLine1	2,094,000	2,910,000	17,700,000	0.68	3.09	2.40
MBZero{0,1,2,3,4,5}	36,000	72,000	6,048,000	0.01	1.05	1.04
DCT_{1,2}	3,360,000	10,320,000	13,488,000	1.10	2.35	1.26
MBGetLine1	1,296,000	1,728,000	12,624,000	0.42	2.20	1.78
Total (ycles)	**306,581,121**	**469,928,828**	**573,521,134**			

APPENDIX B

XML interfaces in ASAM design flow

Following we describe a subset of the XML files used in ASAM design flow: we present the input and output files of the probabilistic and deterministic DSE.

B.1 Input constraints

This XML file contains the input constraints provided by the designer for each of the applications in input. In particular there is the working frequency (in MHz) for the final platform, the maximum area (in μm^2) and the deadline (in μs) of each application. In Listing B.1 there are the input constraints for ECG application.

Listing B.1: Input constraints of ECG application

```
1 <?xml version="1.0" encoding="utf-8" ?>
2
3 <system id="1" area="8000000" power="0" frequency="1">
4     <application id="1" name="ecg" deadline="16000000">
5     </application>
6 </system>
```

B.2 Initial Platform

The initial platform file contains the CP model of the platform expresses as a graph, in which the nodes are PEs or communication elements (CEs). For each CE, there is also the information required to estimate the transmission time of a message. For a bus, we have the information about the width w and frequency f; we can set multiple width ad frequency to allow the macro-architecture DSE to explore them. In Listing B.2 there is CP model with two PEs and a host processor interconnected by a bus.

Listing B.2: Input platform for ECG application

```
<?xml version="1.0" encoding="utf-8" ?>

<platform id="1" addressWidth="32" dataWidth="32" burstSize="4">
<node id="0">
        <name>host</name>
        <type>PE</type>
</node>
<node id="1">
        <name>PE1</name>
        <type>PE</type>
</node>
<node id="2">
        <name>PE2</name>
        <type>PE</type>
</node>
<node id="3">
        <name>CE1</name>
        <type>bus</type>
        <typeBusList>
           <typeBus id="1" width="32" frequency="1"/>
        </typeBusList>
</node>
<edge source="1" target="3" />
<edge source="2" target="3" />
<edge source="0" target="3" />
</platform>
```

The initial platform file is update after the probabilistic DSE, e.g. only the selected bus type is listed and if there are PEs that are not used, they are removed.

B.3 Application model

The application model, both for SDFG and task graph (TG) models, is represented as a set of nodes (i.e. tasks) interconnected by edges (i.e. messages). An example is shown in Listing B.3 (ECG application). Each task contains:

- name and id: identifiers of the task (the name is use to find the corresponding node in the KPN generated by Compaan)
- functionName: the name of the kernel function executed by the task
- lWECT and uWCET: the lower and upper bound for the WCET (C_j^l and C_j^u), respectively
- layer and priority: used as support to the schedulability analysis (the layer is automatically calculated, the priority is assigned according to the task name, but can be modified by the user)
- firing: total number of executions of the task

Each message contains:

- name and id: identifiers of the message (the name is use to find the corresponding edge in the KPN generated by Compaan)
- source and target: id of the source and target tasks that the message connects
- data: size of a token of data (in bit) that it is exchange between the source and target tasks.
- layer and priority: used as support to the schedulability analysis (the layer is automatically calculated, the priority is assigned according to the task name, but can be modified by the user)

Listing B.3: Application model for ECG application annotated with the upper and lower bounds by Phase 1 of micro-architecture DSE (input of the probabilistic DSE)

```
<?xml version="1.0" encoding="utf-8" ?>
<applications>
<application id="">
<node id="8">
    <name>ND_3</name>
    <functionName>highpass</functionName>
    <lWCET id="1">450084.0</lWCET>
    <uWCET id="1">840021.0</uWCET>
    <firing>10000</firing>
    <layer>5</layer>
    <priority>3</priority>
    <type>transformer</type>
</node>
<node id="7">
    <name>ND_4</name>
    <functionName>derivative</functionName>
    <lWCET id="1">10000.0</lWCET>
```

```
18      <uWCET id="1">10000.0</uWCET>
19      <firing>10000</firing>
20      <layer>7</layer>
21      <priority>4</priority>
22      <type>transformer</type>
23  </node>
24  <node id="2">
25      <name>ND_6</name>
26      <functionName>integrative</functionName>
27      <lWCET id="1">1.0830071E7</lWCET>
28      <uWCET id="1">2.0370915E7</uWCET>
29      <firing>10000</firing>
30      <layer>11</layer>
31      <priority>6</priority>
32      <type>transformer</type>
33  </node>
34  <node id="6">
35      <name>ND_8</name>
36      <functionName>printOut</functionName>
37      <lWCET id="1">10000.0</lWCET>
38      <uWCET id="1">10000.0</uWCET>
39      <firing>10000</firing>
40      <layer>15</layer>
41      <priority>8</priority>
42      <type>sink</type>
43  </node>
44  <node id="3">
45      <name>ND_7</name>
46      <functionName>detect</functionName>
47      <lWCET id="1">99614.0</lWCET>
48      <uWCET id="1">155070.0</uWCET>
49      <firing>10000</firing>
50      <layer>13</layer>
51      <priority>7</priority>
52      <type>transformer</type>
53  </node>
54  <node id="5">
55      <name>ND_5</name>
56      <functionName>square</functionName>
57      <lWCET id="1">10000.0</lWCET>
58      <uWCET id="1">10000.0</uWCET>
59      <firing>10000</firing>
60      <layer>9</layer>
61      <priority>5</priority>
62      <type>transformer</type>
63  </node>
64  <node id="4">
65      <name>ND_2</name>
66      <functionName>lowpass</functionName>
67      <lWCET id="1">400015.0</lWCET>
68      <uWCET id="1">450010.0</uWCET>
69      <firing>10000</firing>
70      <layer>3</layer>
71      <priority>2</priority>
72      <type>transformer</type>
```

```
</node>
<node id="1">
    <name>ND_1</name>
    <functionName>get_sample</functionName>
    <lWCET id="1">130003.0</lWCET>
    <uWCET id="1">150004.0</uWCET>
    <firing>10000</firing>
    <layer>1</layer>
    <priority>1</priority>
    <type>source</type>
</node>
<edge id="14" source="7" target="5">
    <name>ED_4</name>
    <data>32</data>
    <layer>8</layer>
    <priority>12</priority>
    <throughput>10000</throughput>
</edge>
<edge id="15" source="3" target="6">
    <name>ED_7_ED_8_ED_9</name>
    <data>32</data>
    <layer>14</layer>
    <priority>15</priority>
    <throughput>10000</throughput>
</edge>
<edge id="12" source="1" target="4">
    <name>ED_1</name>
    <data>32</data>
    <layer>2</layer>
    <priority>9</priority>
    <throughput>10000</throughput>
</edge>
<edge id="9" source="8" target="7">
    <name>ED_3</name>
    <data>32</data>
    <layer>6</layer>
    <priority>11</priority>
    <throughput>10000</throughput>
</edge>
<edge id="10" source="2" target="3">
    <name>ED_6</name>
    <data>32</data>
    <layer>12</layer>
    <priority>14</priority>
    <throughput>10000</throughput>
</edge>
<edge id="13" source="4" target="8">
    <name>ED_2</name>
    <data>32</data>
    <layer>4</layer>
    <priority>10</priority>
    <throughput>10000</throughput>
</edge>
<edge id="11" source="5" target="2">
    <name>ED_5</name>
```

```
128     <data>32</data>
129     <layer>10</layer>
130     <priority>13</priority>
131     <throughput>10000</throughput>
132 </edge>
133 </application>
134 </applications>
```

B.4 Task clustering solution

An XML file is also used to define a task clustering solution. The file containing the solutions is generated by the probabilistic DSE and during the execution of the ASAM flow is update with the information produced by the micro-architecture DSE (Phase 2) and by the deterministic DSE.

B.4.1 Output of probabilistic DSE

The XML file produced as output of the probabilistic DSE contains one or more task clustering solutions. An example is available in Listing B.4 (task clustering solution found for ECG case study). Each solution is described by the XML element <*solution*> and contains a group of clusters (XML element <*cluster*>). A cluster is a group of nodes (i.e. tasks) or edges (i.e. messages). The clusters containing tasks correspond to ASIPs. The attribute $processorId$ of each cluster indicates the ASIP id. The host processor has $processorId = 0$. We cluster together the messages that are associated with the same CE; the messages that are not associated with any CE (messages between tasks clustered together) are also grouped in a dummy cluster identified by the attribute $processorId = -1$.

Listing B.4: Task clustering solution found by the probabilistic DSE (input of Phase 2 of micro-architecture DSE)

```
1 <?xml version="1.0" encoding="utf-8" ?>
2 <solutions>
3 <solution id="1" cost="0.57" proc="3">
4 <cluster id="1:1:0" pathToKPN="./outputs/1_0_kpn.kpn" processorId="
      0">
5 <node id="1:0:1">
6     <name>ND_1</name>
7     <functionName>get_sample</functionName>
8     <lWCET id="1">13.0003</lWCET>
9     <uWCET id="1">15.0004</uWCET>
10    <firing>10000</firing>
11    <layer>1</layer>
```

```
12      <priority>1</priority>
13      <type>source</type>
14  </node>
15  <node id="1:0:6">
16      <name>ND_8</name>
17      <functionName>printOut</functionName>
18      <lWCET id="1">1.0</lWCET>
19      <uWCET id="1">1.0</uWCET>
20      <firing>10000</firing>
21      <layer>15</layer>
22      <priority>8</priority>
23      <type>sink</type>
24  </node>
25  </cluster>
26  <cluster id="1:1:2" pathToKPN="./outputs/1_2_kpn.kpn" processorId="
        2">
27  <node id="1:2:2">
28      <name>ND_6</name>
29      <functionName>integrative</functionName>
30      <lWCET id="1">1083.0071</lWCET>
31      <uWCET id="1">2037.0915</uWCET>
32      <firing>10000</firing>
33      <layer>11</layer>
34      <priority>6</priority>
35      <type>transformer</type>
36  </node>
37  <node id="1:2:3">
38      <name>ND_7</name>
39      <functionName>detect</functionName>
40      <lWCET id="1">9.9614</lWCET>
41      <uWCET id="1">15.507</uWCET>
42      <firing>10000</firing>
43      <layer>13</layer>
44      <priority>7</priority>
45      <type>transformer</type>
46  </node>
47  </cluster>
48  <cluster id="1:1:1" pathToKPN="./outputs/1_1_kpn.kpn" processorId="
        1">
49  <node id="1:1:4">
50      <name>ND_2</name>
51      <functionName>lowpass</functionName>
52      <lWCET id="1">40.0015</lWCET>
53      <uWCET id="1">45.001</uWCET>
54      <firing>10000</firing>
55      <layer>3</layer>
56      <priority>2</priority>
57      <type>transformer</type>
58  </node>
59  <node id="1:1:5">
60      <name>ND_5</name>
61      <functionName>square</functionName>
62      <lWCET id="1">1.0</lWCET>
63      <uWCET id="1">1.0</uWCET>
64      <firing>10000</firing>
```

```
65      <layer>9</layer>
66      <priority>5</priority>
67      <type>transformer</type>
68  </node>
69  <node id="1:1:7">
70      <name>ND_4</name>
71      <functionName>derivative</functionName>
72      <lWCET id="1">1.0</lWCET>
73      <uWCET id="1">1.0</uWCET>
74      <firing>10000</firing>
75      <layer>7</layer>
76      <priority>4</priority>
77      <type>transformer</type>
78  </node>
79  <node id="1:1:8">
80      <name>ND_3</name>
81      <functionName>highpass</functionName>
82      <lWCET id="1">45.0084</lWCET>
83      <uWCET id="1">84.0021</uWCET>
84      <firing>10000</firing>
85      <layer>5</layer>
86      <priority>3</priority>
87      <type>transformer</type>
88  </node>
89  </cluster>
90  <cluster id="1:-1" processorId="-1">
91  <edge id="1:-1:9" source="1:1" target="1:1">
92      <name>ED_3</name>
93      <data>32</data>
94      <priority>11</priority>
95      <layer>6</layer>
96  </edge>
97  <edge id="1:-1:10" source="1:2" target="1:2">
98      <name>ED_6</name>
99      <data>32</data>
100     <priority>14</priority>
101     <layer>12</layer>
102 </edge>
103 <edge id="1:-1:13" source="1:1" target="1:1">
104     <name>ED_2</name>
105     <data>32</data>
106     <priority>10</priority>
107     <layer>4</layer>
108 </edge>
109 <edge id="1:-1:14" source="1:1" target="1:1">
110     <name>ED_4</name>
111     <data>32</data>
112     <priority>12</priority>
113     <layer>8</layer>
114 </edge>
115 <edge id="1:-1:15" source="1:2" target="1:0">
116     <name>ED_7_ED_8_ED_9</name>
117     <data>32</data>
118     <priority>15</priority>
119     <layer>14</layer>
```

```
</edge>
</cluster>
<cluster id="1:3" processorId="3">
<edge id="1:3:11" source="1:1" target="1:2">
    <name>ED_5</name>
    <data>32</data>
    <priority>13</priority>
    <layer>10</layer>
</edge>
<edge id="1:3:12" source="1:0" target="1:1">
    <name>ED_1</name>
    <data>32</data>
    <priority>9</priority>
    <layer>2</layer>
</edge>
</cluster>
</solution>
</solutions>
```

B.4.2 Output of micro-architecture DSE (Phase 2)

Phase 2 of micro-architecture DSE uses the file produced by the probabilistic DSE and annotates it with the performance and area estimation of different micro-architecture configurations. An example is available in Listing B.5 (output of Phase 2 for ECG case study). The area together with other optimization parameters, if available, are specified using the XML element *parametersList*; there is an XML element *parameters* for each cluster and an XML element <*config*> for each of the micro-architecture configuration found by Phase 2. Each XML element <*config*> has *id*, *area* and *power* attributes. Additionally, for each task and micro-architecture, it adds information about the performance of the tasks: i.e. the start and end cycles (with respect to an initial offset=0). The source and sink tasks associated with the host processor are not evaluated by Phase 2; as a consequence there are no estimated start and end times for them.

Listing B.5: Task clustering solution annotated with area and performance for multiple micro-architecture for each ASIP (input of the deterministic DSE)

```
<?xml version="1.0" encoding="utf-8" ?>
<solutions>
<solution id="1">
<cluster id="1:1:0" pathToKPN="./outputs/1_0_kpn.kpn" processorId="
    0">
<node id="1:0:1">
    <name>ND_1</name>
    <functionName>get_sample</functionName>
    <firing>10000</firing>
    <priority>1</priority>
    <layer>1</layer>
</node>
```

```
<node id="1:0:6">
    <name>ND_8</name>
    <functionName>printOut</functionName>
    <firing>10000</firing>
    <priority>8</priority>
    <layer>15</layer>
</node>
</cluster>
<cluster id="1:1:2" pathToKPN="./outputs/1_2_kpn.kpn" processorId="
    2">
<node id="1:2:2">
    <name>ND_6</name>
    <functionName>integrative</functionName>
    <firing>10000</firing>
    <priority>6</priority>
    <layer>11</layer>
    <paretoList id="1">
        <pareto id="0">
            <startPoint>0</startPoint>
            <endPoint>12450000</endPoint>
        </pareto>
    </paretoList>
</node>
<node id="1:2:3">
    <name>ND_7</name>
    <functionName>detect</functionName>
    <firing>10000</firing>
    <priority>7</priority>
    <layer>13</layer>
    <paretoList id="1">
        <pareto id="0">
            <startPoint>12450000</startPoint>
            <endPoint>12559989</endPoint>
        </pareto>
    </paretoList>
</node>
</cluster>
<cluster id="1:1:1" pathToKPN="./outputs/1_1_kpn.kpn" processorId="
    1">
<node id="1:1:4">
    <name>ND_2</name>
    <functionName>lowpass</functionName>
    <firing>10000</firing>
    <priority>2</priority>
    <layer>3</layer>
    <paretoList id="1">
        <pareto id="0">
            <startPoint>0</startPoint>
            <endPoint>115046</endPoint>
        </pareto>
        <pareto id="1">
            <startPoint>0</startPoint>
            <endPoint>1840</endPoint>
        </pareto>
        <pareto id="2">
```

```
65              <startPoint>0</startPoint>
66              <endPoint>3634</endPoint>
67          </pareto>
68          <pareto id="3">
69              <startPoint>0</startPoint>
70              <endPoint>460000</endPoint>
71          </pareto>
72      </paretoList>
73  </node>
74  <node id="1:1:5">
75      <name>ND_5</name>
76      <functionName>square</functionName>
77      <firing>10000</firing>
78      <priority>5</priority>
79      <layer>9</layer>
80      <paretoList id="1">
81          <pareto id="0">
82              <startPoint>644993</startPoint>
83              <endPoint>654992</endPoint>
84          </pareto>
85          <pareto id="1">
86              <startPoint>531787</startPoint>
87              <endPoint>541786</endPoint>
88          </pareto>
89          <pareto id="2">
90              <startPoint>17689</startPoint>
91              <endPoint>27688</endPoint>
92          </pareto>
93          <pareto id="3">
94              <startPoint>989947</startPoint>
95              <endPoint>992447</endPoint>
96          </pareto>
97      </paretoList>
98  </node>
99  <node id="1:1:7">
100     <name>ND_4</name>
101     <functionName>derivative</functionName>
102     <firing>10000</firing>
103     <priority>4</priority>
104     <layer>7</layer>
105     <paretoList id="1">
106         <pareto id="0">
107             <startPoint>634994</startPoint>
108             <endPoint>644993</endPoint>
109         </pareto>
110         <pareto id="1">
111             <startPoint>521788</startPoint>
112             <endPoint>531787</endPoint>
113         </pareto>
114         <pareto id="2">
115             <startPoint>7690</startPoint>
116             <endPoint>17689</endPoint>
117         </pareto>
118         <pareto id="3">
119             <startPoint>979948</startPoint>
```

```
120             <endPoint>989947</endPoint>
121         </pareto>
122     </paretoList>
123 </node>
124 <node id="1:1:8">
125     <name>ND_3</name>
126     <functionName>highpass</functionName>
127     <firing>10000</firing>
128     <priority>3</priority>
129     <layer>5</layer>
130     <paretoList id="1">
131         <pareto id="0">
132             <startPoint>115046</startPoint>
133             <endPoint>634994</endPoint>
134         </pareto>
135         <pareto id="1">
136             <startPoint>1840</startPoint>
137             <endPoint>521788</endPoint>
138         </pareto>
139         <pareto id="2">
140             <startPoint>3634</startPoint>
141             <endPoint>7690</endPoint>
142         </pareto>
143         <pareto id="3">
144             <startPoint>460000</startPoint>
145             <endPoint>979948</endPoint>
146         </pareto>
147     </paretoList>
148 </node>
149 </cluster>
150 <cluster id="1:-1" processorId="-1">
151 <edge id="1:-1:9" source="1:1" target="1:1">
152     <name>ED_3</name>
153     <data>32</data>
154     <priority>11</priority>
155     <layer>6</layer>
156 </edge>
157 <edge id="1:-1:10" source="1:2" target="1:2">
158     <name>ED_6</name>
159     <data>32</data>
160     <priority>14</priority>
161     <layer>12</layer>
162 </edge>
163 <edge id="1:-1:13" source="1:1" target="1:1">
164     <name>ED_2</name>
165     <data>32</data>
166     <priority>10</priority>
167     <layer>4</layer>
168 </edge>
169 <edge id="1:-1:14" source="1:1" target="1:1">
170     <name>ED_4</name>
171     <data>32</data>
172     <priority>12</priority>
173     <layer>8</layer>
174 </edge>
```

```
<edge id="1:-1:15" source="1:2" target="1:0">
    <name>ED_7_ED_8_ED_9</name>
    <data>32</data>
    <priority>15</priority>
    <layer>14</layer>
</edge>
</cluster>
<cluster id="1:3" processorId="3">
<edge id="1:3:11" source="1:1" target="1:2">
    <name>ED_5</name>
    <data>32</data>
    <priority>13</priority>
    <layer>10</layer>
</edge>
<edge id="1:3:12" source="1:0" target="1:1">
    <name>ED_1</name>
    <data>32</data>
    <priority>9</priority>
    <layer>2</layer>
</edge>
</cluster>
<parametersList>
    <parameters clusterId="1:1:2">
        <config id="0" area="3882879.25" power="-1"/>
    </parameters>
    <parameters clusterId="1:1:1">
        <config id="0" area="3885215.25" power="-1"/>
        <config id="1" area="3910444.0" power="-1"/>
        <config id="2" area="5149420.0" power="-1"/>
        <config id="3" area="3884280.75" power="-1"/>
    </parameters>
</parametersList>
</solution>
</solutions>
```

B.4.3 Output of deterministic DSE

The deterministic DSE performs a schedulability analysis of the application executing on the multi-ASIP platform together with an evaluation of the total area of the platform. It generates an XML file containing the task clustering solution: the information produced by the micro-architecture DSE is removed, as it has already been processed and the total WCET of the application and the platform area are added as attributes to the XML elements <*solution*>. Moreover, for each XML element <*cluster*>, there is an attribute *config* that contains the *id* of the selected micro-architecture configuration. The output for the deterministic DSE for the ECG case study is shown in Listing B.6.

Listing B.6: Task clustering with selected micro-architecture for each ASIP and total system area and performance

```
<?xml version="1.0" encoding="utf-8" ?>
<solutions deadline="1.6E7" maxArea="8450000.0" maxPower="100.0">
<solution id="2" wcet="1.2550167E7" area="7767160.0" power="-2.0">
<cluster id="1:0" config="0" processorId="0">
<node id="1">
    <name>ND_1</name>
    <functionName>get_sample</functionName>
    <priority>1</priority>
</node>
<node id="6">
    <name>ND_8</name>
    <functionName>printOut</functionName>
    <priority>8</priority>
</node>
</cluster>
<cluster id="1:2" config="0" processorId="2">
<node id="2">
    <name>ND_6</name>
    <functionName>integrative</functionName>
    <priority>6</priority>
</node>
<node id="3">
    <name>ND_7</name>
    <functionName>detect</functionName>
    <priority>7</priority>
</node>
</cluster>
<cluster id="1:1" config="4" processorId="1">
<node id="4">
    <name>ND_2</name>
    <functionName>lowpass</functionName>
    <priority>2</priority>
</node>
<node id="5">
    <name>ND_5</name>
    <functionName>square</functionName>
    <priority>5</priority>
</node>
<node id="7">
    <name>ND_4</name>
    <functionName>derivative</functionName>
    <priority>4</priority>
</node>
<node id="8">
    <name>ND_3</name>
    <functionName>highpass</functionName>
    <priority>3</priority>
</node>
</cluster>
<cluster id="1:-1" config="-1" processorId="-1">
<edge id="9" source="1" target="1">
    <name>ED_3</name>
    <data>32</data>
    <priority>11</priority>
    <bus>-1</bus>
```

```
</edge>
<edge id="10" source="2" target="2">
    <name>ED_6</name>
    <data>32</data>
    <priority>14</priority>
    <bus>-1</bus>
</edge>
<edge id="13" source="1" target="1">
    <name>ED_2</name>
    <data>32</data>
    <priority>10</priority>
    <bus>-1</bus>
</edge>
<edge id="14" source="1" target="1">
    <name>ED_4</name>
    <data>32</data>
    <priority>12</priority>
    <bus>-1</bus>
</edge>
<edge id="15" source="2" target="0">
    <name>ED_7_ED_8_ED_9</name>
    <data>32</data>
    <priority>15</priority>
    <bus>-1</bus>
</edge>
</cluster>
<cluster id="1:3" config="0" processorId="3">
<edge id="11" source="1" target="2">
    <name>ED_5</name>
    <data>32</data>
    <priority>13</priority>
    <bus>3</bus>
</edge>
<edge id="12" source="0" target="1">
    <name>ED_1</name>
    <data>32</data>
    <priority>9</priority>
    <bus>3</bus>
</edge>
</cluster>
</solution>
</solutions>
```

Bibliography

[1] Compaan compiler. http://www.compaandesign.com, June 2014.

[2] E3S Benchmark. http://ziyang.eecs.umich.edu/~dickrp/e3s/, June 2014.

[3] MAD, MPEG Audio Decoder. http://www.underbit.com/products/mad/, June 2014.

[4] Metaheuristic Algorithms in Java, jMetal library. http://jmetal.sourceforge.net, July 2014.

[5] TriMedia TM-1300 datasheet. http://www.datasheetcatalog.org/datasheet/philips/PTM1300.pdf, June 2014.

[6] P. Agrawal, P. Raghavan, M. Hartman, N. Sharma, L. Van der Perre, and F. Catthoor. Early exploration for platform architecture instantiation with multi-mode application partitioning. In *Proceedings of the 50th Design Automation Conference*, pages 1–8, 2013.

[7] ASAM. Automatic architecture synthesis and application mapping. http://www.asam-project.org, June 2014.

[8] J. Axelsson. A method for evaluating uncertainties in the early development phases of embedded real-time systems. In *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 72–75. IEEE Computer Society, 2005.

[9] A. Baghdadi, D. Lyonnard, N. Zergainoh, and A. Jerraya. An efficient architecture model for systematic design of application-specific multiprocessor soc. In

Proceedings of the Conference on Design, automation and test in Europe, pages 55–63. IEEE Press, 2001.

[10] M.-W. Benabderrahmane, L.-N. Pouchet, A. Cohen, and C. Bastoul. The polyhedral model is more widely applicable than you think. In *Compiler Construction*, pages 283–303. Springer, 2010.

[11] L. Benini, D. Bertozzi, A. Bogliolo, F. Menichelli, and M. Olivieri. Mparm: Exploring the multi-processor soc design space with systemc. *Journal of VLSI signal processing systems for signal, image and video technology*, 41(2):169–182, 2005.

[12] M. Berekovic. rASIP: reconfigurable application specific instruction set processors - slide set. `http://www.dagstuhl.de/Materials/Files/10/10281/10281.BerekovicMladen.Slides.pdf`, July 2014.

[13] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini, and G. De Micheli. NoC synthesis flow for customized domain specific multiprocessor systems-on-chip. *IEEE Transactions on Parallel and Distributed Systems*, 16(2):113–129, Feb. 2005.

[14] C. Brehm, T. Ilnseher, and N. Wehn. A scalable multi-ASIP architecture for standard compliant trellis decoding. In *International SoC Design Conference*, pages 349–352. IEEE, Nov. 2011.

[15] E. Burke and G. Kendall. *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, chapter Introduction, Chapter 1, pages 97–125. Springer, 2005.

[16] D. G. Chinnery and K. Keutzer. *Closing the Power Gap between ASIC and Custom - Tools and Techniques for Low Power Design.* Springer, 2007.

[17] R. J. Cole, B. M. Maggs, and R. K. Sitaraman. On the benefit of supporting virtual channels in wormhole routers. *Journal of Computer and System Sciences*, 62(1):152 – 177, 2001.

[18] M. Damavandpeyma, S. Stuijk, T. Basten, M. Geilen, and H. Corporaal. Schedule-extended synchronous dataflow graphs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(10):1495–1508, Oct. 2013.

[19] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. *Lecture notes in computer science*, 1917:849–858, 2000.

[20] M. Dion and Y. Robert. Mapping affine loop nests. *Parallel Computing*, 22, 1996.

[21] H. C. Doan, H. Javaid, and S. Parameswaran. Multi-ASIP based parallel and scalable implementation of motion estimation kernel for high definition videos. In *2011 9th IEEE Symposium on Embedded Systems for Real-Time Multimedia*, pages 56–65. IEEE, Oct. 2011.

[22] S. F. Edgar. Estimation of worst-case execution time using statistical analysis. *University OF York Department of Computer Science-publications-YCST*, 2004.

[23] C. Erbas, A. D. Pimentel, M. Thompson, and S. Polstra. A framework for system-level modeling and simulation of embedded systems architectures. *EURASIP Journal of Embedded Systems*, 2007(1):1–11, Jan. 2007.

[24] F. Ferrandi, P.-L. Lanzi, C. Pilato, D. Sciuto, and A. Tumeo. Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(6):911–924, June 2010.

[25] J. A. Fisher, P. Faraboschi, and C. Young. *Embedded computing - a VLIW approach to architecture, compilers, and tools.* Morgan Kaufmann, 2005.

[26] J. A. Fisher, P. Faraboschi, and C. Young. VEX, a VLIW Example. `http://www.hpl.hp.com/downloads/vex/`, June 2014.

[27] D. K. Frank Ieromnimon. Application of the MOSART Flow on the WiMAX (802.16e) PHY Layer, 2013.

[28] D. Gangadharan, L. Micconi, P. Pop, and J. Madsen. Multi-ASIP platform synthesis for event-triggered applications with cost/performance trade-offs. In *Proceedings of Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 277–286, Aug. 2013.

[29] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[30] A. Geven. Mixed criticality for complex networked systems. `http://cordis.europa.eu/fp7/ict/embedded-systems-engineering/presentations/geven.pdf`, July 2014.

[31] A.-H. Ghamarian, M. C. W. Geilen, S. Stuijk, T. Basten, A. J. M. Moonen, M. Bekooij, B. Theelen, and M. Mousavi. Throughput analysis of synchronous data flow graphs. In *Proceedings of the 6th International Conference on Application of Concurrency to System Design*, pages 25–36, June 2006.

[32] C. Glitia, P. Boulet, E. Lenormand, and M. Barreteau. Repetitive model refactoring strategy for the design space exploration of intensive signal processing applications. *Journal of Systems Architecture*, 57(9):815–829, 2011.

[33] D. Goodwin and D. Petkov. Automatic generation of application specific processors. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, CASES '03, pages 137–147. ACM, 2003.

[34] Z. Guo, B. Buyukkurt, W. Najjar, and K. Vissers. Optimized generation of datapath from c codes for fpgas. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, DATE '05, pages 112–117. IEEE Computer Society, 2005.

[35] J. C. P. Gutiérrez and M. G. Harbour. Schedulability analysis for tasks with static and dynamic offsets. In *RTSS*, pages 26–37. IEEE Computer Society, 1998.

[36] J. Hansen, S. A. Hissam, and G. A. Moreno. Statistical-based WCET estimation and validation. In *Proceedings of the 9th International Workshop on Worst-Case Execution Time Analysis*, pages 1–11, 2009.

[37] A. Hansson, M. Subburaman, and K. Goossens. Aelite: A flit-synchronous Network on Chip with composable and predictable services. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, DATE '09, pages 250–255. European Design and Automation Association, Apr. 2009.

[38] P.-K. Huang, M. Hashemi, and S. Ghiasi. System-level performance estimation for application-specific mpsoc interconnect synthesis. In *Symposium on Application Specific Processors*, pages 95–100, June 2008.

[39] P.-K. Huang, M. Hashemi, and S. Ghiasi. System-level performance estimation for application-specific MPSoC interconnect synthesis. In *Symposium on Application Specific Processors*, pages 95–100, June 2008.

[40] M. Jain, M. Balakrishnan, and A. Kumar. ASIP design methodologies: survey and issues. In *14th International Conference on VLSI Design*, pages 76–81. IEEE Comput. Soc, Jan. 2001.

[41] H. Javaid and S. Parameswaran. Synthesis of heterogeneous pipelined multiprocessor systems using ILP. In *Proceedings of the 6th IEEE/ACM/IFIP international Conference on Hardware/Software codesign and system synthesis*, pages 1–6. ACM Press, Oct. 2008.

[42] R. Jordans, R. Corvino, L. Jozwiak, and H. Corporaal. Exploring processor parallelism: Estimation methods and optimization strategies. In *IEEE 16th International Symposium on Design and Diagnostics of Electronic Circuits Systems*, pages 18–23, Apr. 2013.

[43] L. Jozwiak, M. Lindwer, R. Corvino, P. Meloni, L. Micconi, J. Madsen, E. Diken, D. Gangadharan, R. Jordans, S. Pomata, P. Pop, G. Tuveri, L. Raffo, and G. Notarangelo. ASAM: Automatic architecture synthesis and application mapping. *Microprocessors and Microsystems*, 37(8):1002–1019, Nov. 2013.

[44] T. Kangas, P. Kukkala, H. Orsila, E. Salminen, M. Hännikäinen, T. D. Hämäläinen, J. Riihimäki, and K. Kuusilinna. UML-based multiprocessor SoC design framework. *ACM Transactions on Embedded Computing Systems*, 5(2):281–320, May 2006.

[45] P. Karlstrom, W. Zhou, C.-h. Wang, and D. Liu. Design of PIONEER: A case study using NoGap. In *Proceedings of the Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics (PrimeAsia)*, pages 53–56. IEEE, 2010.

[46] K. Karuri, R. Leupers, G. Ascheid, and H. Meyr. A generic design flow for application specific processor customization through instruction-set extensions (ises). In *Embedded Computer Systems: Architectures, Modeling, and Simulation*, volume 5657 of *Lecture Notes in Computer Science*, pages 204–214. Springer Berlin Heidelberg, 2009.

[47] T. Kempf, M. Doerper, R. Leupers, G. Ascheid, H. Meyr, T. Kogel, and B. Vanthournout. A modular simulation framework for spatial and temporal task mapping onto multi-processor soc platforms. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, DATE '05, pages 876–881. IEEE Computer Society, 2005.

[48] K. Kennedy and K. S. McKinley. *Maximizing loop parallelism and improving data locality via loop fusion and distribution*. Springer, 1994.

[49] K. Keutzer, A. Newton, J. Rabaey, and A. Sangiovanni-Vincentelli. System-level design: Orthogonalization of concerns and platform-based design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(12):1523–1543, 2000.

[50] B. Kienhuis, E. Rijpkema, and E. Deprettere. Compaan. In *Proceedings of the eighth international workshop on Hardware/software codesign - CODES '00*, pages 13–17. ACM Press, May 2000.

[51] H. Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1st edition, 1997.

[52] J. Kreku, M. Hoppari, T. Kestilä, Y. Qu, J.-P. Soininen, P. Andersson, and K. Tiensyrjä. Combining uml2 application and systemc platform modelling for performance evaluation of real-time embedded systems. *EURASIP Journal of Embedded Systems*, 2008:6:1–6:18, Jan. 2008.

[53] V. Krishnan and S. Katkoori. A genetic algorithm for the design space exploration of datapaths during high-level synthesis. *IEEE Transactions on Evolutionary Computation*, 10(3):213–229, June 2006.

[54] A. Kumar, S. Fernando, Y. Ha, B. Mesman, and H. Corporaal. Multiprocessor systems synthesis for multiple use-cases of multiple applications on FPGA. *ACM Transactions on Design Automation of Electronic Systems*, 13(3):1–27, July 2008.

[55] A. Kumar, A. Hansson, J. Huisken, and H. Corporaal. Interactive presentation: An fpga design flow for reconfigurable network-based multi-processor systems on chip. In *Proceedings of the Conference on Design, automation and test in Europe*, pages 117–122, 2007.

[56] Y.-K. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, 31(4):406–471, Dec. 1999.

[57] C. Lee, S. Kim, and S. Ha. A systematic design space exploration of mpsoc based on synchronous data flow specification. *Journal of Signal Processing Systems*, 58(2):193–213, 2010.

[58] E. A. Lee and D. G. Messerschmitt. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. Comput.*, 36(1):24–35, Jan. 1987.

[59] T. Lei and S. Kumar. A two-step genetic algorithm for mapping task graphs to a network on chip architecture. In *Proceedings of the Euromicro Symposium on Digital System Design*, pages 180–187, Sept. 2003.

[60] Leiden University. MJPEG code. `http://www.artist-embedded.org/artist/Benchmarks.html`, June 2014.

[61] Y. A. Li and J. K. Antonio. Estimating the execution time distribution for a task graph in a heterogeneous computing system. In *Proceedings of the 6th Heterogeneous Computing Workshop*, pages 172–184, 1997.

[62] M. Lindwer. The future of data-parallel embedded systems, keynote speach. `http://dsmc2.eap.gr/dsd2011/docs/DSD2011_Keynote_Speaker3.pdffiles`, July 2014.

[63] M. Lindwer, L. Jóźwiak, J. Madsen, B. Kienhuis, P. Meloni, L. Raffo, and G. Notarangelo. Initial design methodology, flow, and tool requirements. Deliverable 1.1, ASAM Project, Nov. 2010.

[64] LLVM. The compiler infrastructure. `http://llvm.org`, June 2014.

[65] Z. Lu and A. Jantsch. Tdm virtual-circuit configuration for network-on-chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(8):1021–1034, Aug. 2008.

[66] D. Lyonnard, S. Yoo, A. Baghdadi, and A. Jerraya. Automatic generation of application-specific architectures for heterogeneous multiprocessor system-on-chip. In *Proceedings of the Design Automation Conference*, pages 518–523, 2001.

[67] V. V. Mark Ewert, Prakash Iyer. Hotchips 2013: Clovertrail+ smartphone soc platform; slide set. `http://www.hotchips.org/wp-content/uploads/hc_archives/hc25/HC25.10-SoC1-epub/HC25.26.130-Clovertrail-Smartphone-Iyer-Intel.pdf`, July 2014.

[68] H. Meyr, O. Schliebusch, A. Wieferink, D. Kammler, E. Witte, O. Lüthje, M. Hohenauer, G. Braun, and A. Chattopadhyay. Designing and modeling MPSoC processors and communication architectures. In *Building ASIPS: The Mescal Methodology*, pages 229–280. Springer US, 2005.

[69] L. Micconi, D. Gangadharan, P. Pop, and J. Madsen. Multi-ASIP platform synthesis for real-time applications. In *2013 8th IEEE International Symposium on Industrial Embedded Systems*, pages 59–67. IEEE, June 2013.

[70] L. Micconi, J. Madsen, and P. Pop. An Uncertainty Model for System-Level design of Multi-ASIP Platforms. *Under revision*.

[71] N. Moreano, E. Borin, C. D. Souza, and G. Araujo. Efficient datapath merging for partially reconfigurable architectures. In *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, pages 969–980, 2005.

[72] A. Morgan, H. Elmiligi, M. El-Kharashi, and F. Gebali. Multi-objective optimization of noc standard architectures using genetic algorithms. In *IEEE International Symposium on Signal Processing and Information Technology*, pages 85–90, Dec. 2010.

[73] R. Muhammad, L. Apvrille, and R. Pacalet. Evaluation of ASIPs design with LISATek. In M. Berekovic, N. J. Dimopoulos, and S. Wong, editors, *SAMOS*, Lecture Notes in Computer Science, pages 177–186. Springer, 2008.

[74] O. Muller, A. Baghdadi, and M. Jezequel. From Parallelism Levels to a Multi-ASIP Architecture for Turbo Decoding. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(1):92–102, Jan. 2009.

[75] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2):62–76, 1993.

[76] M. Nicola, G. Masera, M. Zamboni, H. Ishebabi, D. Kammler, G. Ascheid, and H. Meyr. FFT processor: a case study in ASIP development. In *Proceedings of the IST Mobile & Wireless Communications Summit*, 2005.

[77] H. Nikolov, T. Stefanov, and E. Deprettere. Systematic and Automated Multiprocessor System Design, Programming, and Implementation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(3):542–555, Mar. 2008.

[78] H. Nikolov, M. Thompson, T. Stefanov, A. D. Pimentel, S. Polstra, R. Bose, C. Zissulescu, and E. F. Deprettere. Daedalus: toward composable multimedia mp-soc design. In L. Fix, editor, *DAC*, pages 574–579. ACM, 2008.

[79] A. Nohl, F. Schirrmeister, and D. Taussig. Application specific processor design architectures, design methods and tools. In *Proceedings of the International Conference on Computer-Aided Design*, pages 349–352, 2010.

[80] P. G. Paulin, J. P. Knight, and E. F. Girczyc. Hal: A multi-paradigm approach to automatic data path synthesis. In *Proceedings of the 23rd ACM/IEEE Design Automation Conference*, DAC '86, pages 263–270. IEEE Press, 1986.

[81] T. Pop, P. Eles, and Z. Peng. Holistic scheduling and analysis of mixed time/event-triggered distributed embedded systems. In *Proceedings of the 10th International Symp. on HW/SW Codesign*, pages 187–192, 2002.

[82] A. Sangiovanni-Vincentelli. Defining platform-based design. *EEDesign of EE-Times*, 2002.

[83] S. Saponara, L. Fanucci, S. Marsi, and G. Ramponi. Algorithmic and architectural design for real-time and power-efficient Retinex image/video processing. *Journal of Real-Time Image Processing*, 1(4):267–283, May 2007.

[84] M. Schoeberl, F. Brandner, J. Sparsø, and E. Kasapaki. A statically scheduled time-division-multiplexed network-on-chip for real-time systems. In *Sixth IEEE/ACM International Symposium on Networks on Chip*, pages 152–160. IEEE, 2012.

[85] S. L. Shee and S. Parameswaran. Design Methodology for Pipelined Heterogeneous Multiprocessor System. In *Proceedings of the 44th Design Automation Conference*, pages 811–816, 2007.

[86] A. K. Singh, T. Srikanthan, A. Kumar, and W. Jigang. Communication-aware heuristics for run-time task mapping on NoC-based {MPSoC} platforms. *Journal of Systems Architecture*, 56(7):242 – 255, 2010. Special Issue on HW/SW Co-Design: Systems and Networks on Chip.

[87] O. Sinnen. *Task Scheduling for Parallel Systems (Wiley Series on Parallel and Distributed Computing)*. Wiley-Interscience, 2007.

[88] C. C. d. Souza, A. M. Lima, G. Araujo, and N. B. Moreano. The datapath merging problem in reconfigurable systems: Complexity, dual bounds and heuristic evaluation. *J. Exp. Algorithmics*, 10, Dec. 2005.

[89] R. Stefan, A. Molnos, A. Ambrose, and K. Goossens. A TDM NoC supporting QoS, multicast, and fast connection set-up. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, DATE '12, pages 1283–1288. EDA Consortium, 2012.

[90] T. Stefanov, C. Zissulescu, A. Turjan, B. Kienhuis, and E. Deprette. System design using Khan process networks: the Compaan/Laura approach. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, volume 1, pages 340–345. IEEE, 2004.

[91] STMicroelectronics. MPEG4 application. `http://www.st.com`, June 2014.

[92] Synopsys. Synopsys Processor Designer. `http://www.synopsys.com/IP/ProcessorIP/asip/processor-designer/Pages/default.aspx`, July 2014.

[93] Technical University of Denmark. ECG application. Please contact the author of the thesis to get a copy of the C code, June 2014.

[94] Technical University of Eindhoven. MAMPS project, Partitioned JPEG decoder algorithm. `http://www.es.ele.tue.nl/mamps/example.php`, June 2014.

[95] Tensilica. tool-chain for XTensa processor. `http://ip.cadence.com/ipportfolio/tensilica-ip`, July 2014.

[96] R. Walker and S. Chaudhuri. Introduction to the scheduling problem. *Design Test of Computers, IEEE*, 12(2):60 –69, 1995.

[97] F. Wang, C. Nicopoulos, X. Wu, Y. Xie, and N. Vijaykrishnan. Variation-aware task allocation and scheduling for mpsoc. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pages 598–603, Nov. 2007.

[98] G. Wang, W. Gong, and R. Kastner. Application partitioning on programmable platforms using the ant colony optimization. *Journal of Embedded Computing*, 2(1):119–136, 2006.

[99] A. Wieferink, T. Kogel, R. Leupers, G. Ascheid, H. Meyr, G. Braun, and A. Nohl. A system level processor/communication co-exploration methodology for multi-processor system-on-chip platforms. In *Proceedings of the Design Automation Conference*, 2004.

[100] J. Yan and W. Zhang. A time-predictable VLIW processor and its compiler support. *Real-Time Systems*, 38(1):67–84, 2008.

[101] Y. Yassin, P. Kjeldsberg, J. Hulzink, I. Romero, and J. Huisken. Ultra low power application specific instruction-set processor design for a cardiac beat detector algorithm. In *Proceedings of the NORCHIP Conference*, pages 1–4, Nov. 2009.

[102] V. Zaccaria, G. Palermo, F. C. P. di Milano), and G. M. (USI). Multicube Explorer. `http://home.dei.polimi.it/zaccaria/multicube_explorer_v1/Home.html`, June 2014.

[103] H. Zhang. Service disciplines for guaranteed performance service in packet-switching networks. In *Proceedings of the IEEE*, pages 1374–1396, 1995.

[104] W. Zhang, L. Hou, J. Wang, S. Geng, and W. Wu. Comparison research between XY and Odd-Even routing algorithm of a 2-dimension 3x3 Mesh topology Network-on-Chip. In *WRI Global Congress on Intelligent Systems*, volume 3, pages 329–333, May 2009.

[105] C. Zimmer and F. Mueller. Nocmsg: Scalable NoC-based message passing. In *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 186–195. IEEE, 2014.

[106] M. Zuluaga and N. Topham. Resource sharing in custom instruction set extensions. In *Symposium on Application Specific Processors*, pages 7–13, June 2008.

[107] M. Zuluaga and N. Topham. Design-space exploration of resource-sharing solutions for custom instruction set extensions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(12):1788–1801, Dec. 2009.